TAU Performance System®

Sameer Shende
sameer@paratools.com

ParaTools, Inc and University of Oregon.

May 23, 2013, Fermilab, Batavia, IL

Download slides from:

http://tau.uoregon.edu/tau_fnal14.ppt

TAU Performance System®

- **Tuning and Analysis Utilities (18+ year project)**
- **Comprehensive performance profiling and tracing**
  - Integrated, scalable, flexible, portable
  - Targets all parallel programming/execution paradigms
- **Integrated performance toolkit**
  - Instrumentation, measurement, analysis, visualization
  - Widely-ported performance profiling / tracing system
  - Performance data management and data mining
  - Open source (BSD-style license)
- **Easy to integrate in application frameworks**
- ***http://tau.uoregon.edu***

What is TAU?

- **TAU is a performance evaluation tool**
- **It supports parallel profiling and tracing**
- **Profiling shows you how much (total) time was spent in each routine**
- **Tracing shows you *when* the events take place in each process along a timeline**
- **Profiling and tracing can measure time as well as hardware performance counters (cache misses, instructions) from your CPU**
- **TAU can automatically instrument your source code using a package called PDT for routines, loops, I/O, memory, phases, etc.**
- **TAU runs on most HPC platforms and it is free (BSD style license)**
- **TAU has instrumentation, measurement and analysis tools**
  - paraprof is TAU's 3D profile browser
- **To use TAU's automatic source instrumentation, you may set a couple of environment variables and substitute the name of your compiler with a TAU shell script**

Understanding Application Performance using TAU

- How much time is spent in each application routine and outer *loops*? Within loops, what is the contribution of each *statement*?
- How many instructions are executed in these code regions? Floating point, Level 1 and 2 *data cache misses*, hits, branches taken?
- What is the *peak heap memory* usage of the code? When and where is memory allocated/de-allocated? Are there any memory leaks?
- How much time does the application spend performing *I/O*? What is the peak read and write *bandwidth* of individual calls, total volume?
- What is the contribution of different *phases* of the program? What is the time wasted/spent waiting for collectives, and I/O operations in Initialization, Computation, I/O phases?
- How does the application *scale*? What is the efficiency, runtime breakdown of performance across different core counts?

New Features

New Features in TAU

- *tau_exec*: A tool to simplify TAU's usage
- PDT: New parsers from ROSE Compiler
- Power profiling in TAU
- Support for accelerators: OpenCL, Intel Xeon Phi Co-processor
- Event based sampling
- OpenMP instrumentation using OpenMP Tools Interface with Intel compilers (OMPT)
- Support for Intel TBB, CILK, and MPC [paratools.com/mpc]
- Binary rewriting with MAQAO (Intel Exascale Labs, UVSQ) (in Beta)
- ParaProf 3D topology displays
- TAUdb using H2 database

Simplifying the use of TAU!

Uninstrumented code:

- % ssh phi1
- % source /phihome/sameer/pkgs/tau.x86_64.bashrc(or tau.cshrc)
- % mpicc app.c
- % mpirun –np 16 ./a.out

With TAU:

- % mpirun –np 16 tau_exec ./a.out
- % paraprof
- For more Information:
- % mpirun –np 16 tau_exec –ebs ./a.out (or use TAU_SAMPLING=1)
- % tau_rewrite a.out a.inst; mpirun –np 16 ./a.inst (beta)
- % pprof (or paraprof for GUI)

What Can TAU Do?

- **Profiling and tracing**
  - **Profiling** shows you **how much** (total) time was spent in each routine
  - **Tracing** shows you **when** the events take place on a timeline

- **Multi-language debugging**
  - Identify the source location of a crash by unwinding the system callstack
  - Identify memory errors (off-by-one, etc.)

- Profiling and tracing can measure **time** as well as **hardware performance counters** (cache misses, instructions) from your CPU
- TAU can **automatically instrument** your source code using a package called PDT for routines, loops, I/O, memory, phases, etc.
- TAU runs on **all HPC platforms** and it is free (BSD style license)
- TAU includes instrumentation, measurement and analysis tools

What does TAU support?

OpenCL

C/C++

CUDA

UPC

Python

GPI

Fortran

OpenACC

Java

MPI

pthreads

Intel Xeon Phi

OpenMP

Intel

GNU

Co-processor

Sun

MinGW

LLVM

PGI

Cray

AIX

Linux

Windows

Insert yours here

BlueGene

Fujitsu

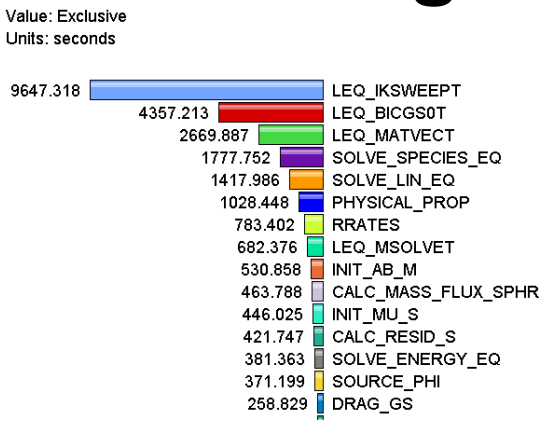
CILK

Intel TBB

OS X

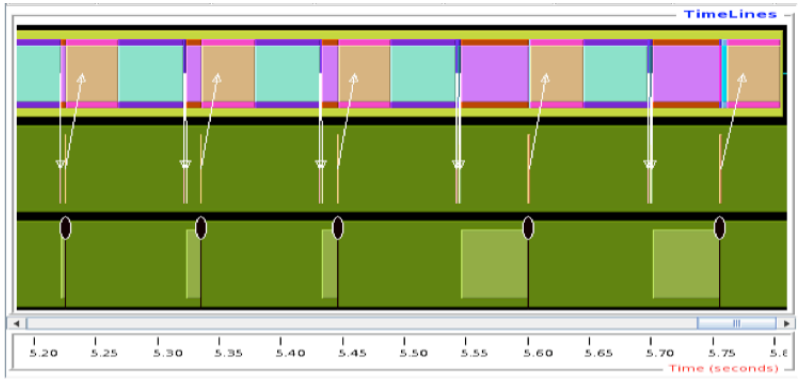
Profiling and Tracing

Profiling



- **Profiling** shows you **how much** (total) time was spent in each routine

Tracing

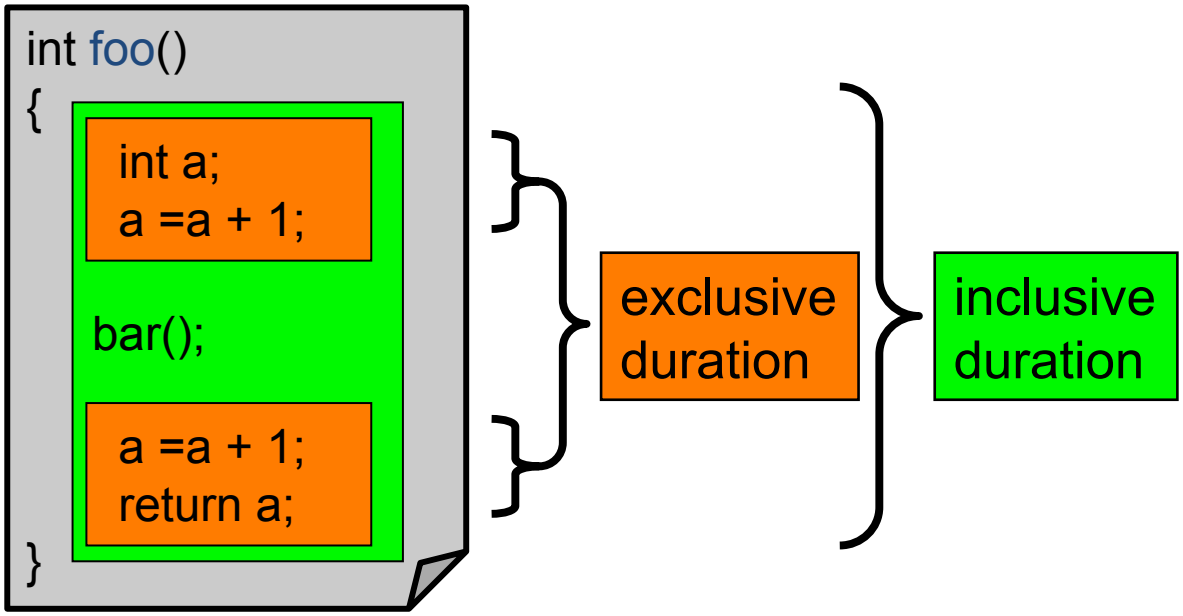


- **Tracing** shows you **when** the events take place on a timeline

- Metrics can be time or hardware performance counters (cache misses, instructions)
- TAU can automatically instrument your source code using a package called PDT for routines, loops, I/O, memory, phases, etc.

Inclusive vs. Exclusive Measurements

- Performance with respect to code regions
- Exclusive measurements for region only
- Inclusive measurements includes child regions



Identifying Potential Bottlenecks

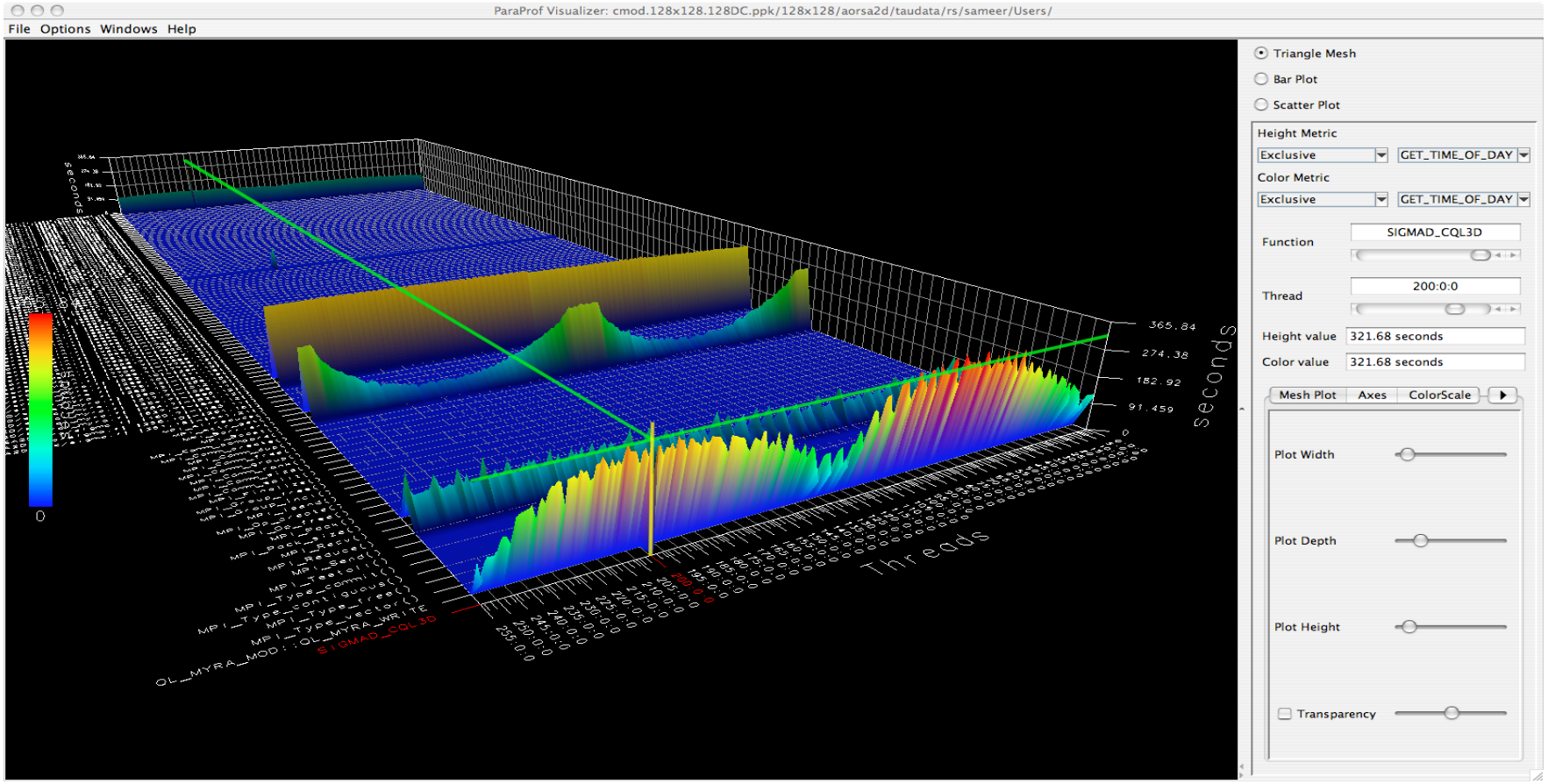
TAU: ParaProf: Mean Data – s3d.loops.12000.ppk

Metric: PAPI_FP_INS / LINUX_TIMERS
Value: Exclusive
Units: Derived metric shown in microseconds format

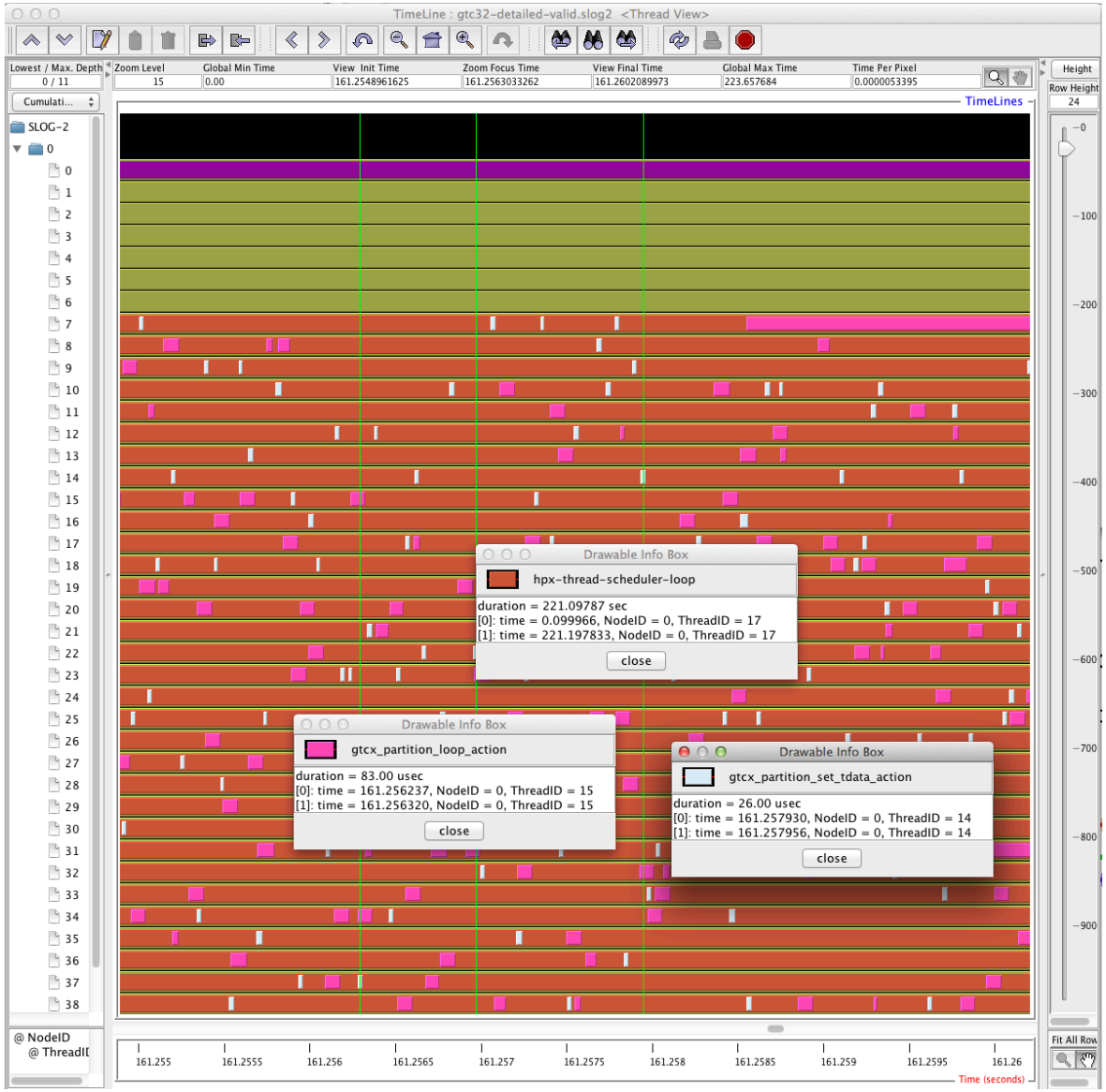
1142.273	Loop: CHEMKIN_M::REACTION_RATE_VEC [{chemkin_m.pp.f90}{457,3}-{471,8}]
884.23	Loop: TRANSPORT_M::COMPUTECOEFFICIENTS [{mixavg_transport_m.pp.f90}{492,5}-{520,9}]
0.014	MPI_Wait()
121.843	Loop: TRANSPORT_M::COMPUTESPECIESDIFFFLUX [{mixavg_transport_m.pp.f90}{630,5}-{656,19}]
0.021	MPI_Isend()
0.06	Loop: RHSF [{rhsf.pp.f90}{209,3}-{211,7}]
183.176	Loop: INTEGRATE [{integrate_erk.pp.f90}{73,3}-{93,13}]
0	MPI_Comm_split()
0	MPI_Barrier()
148.608	RHSF
269.897	Loop: TRANSPORT_M::COMPUTEHEATFLUX [{mixavg_transport_m.pp.f90}{782,5}-{790,19}]
742.83	Loop: THERMCHEM_M::CALC_TEMP [{thermchem_m.pp.f90}{175,5}-{216,9}]
163.317	Loop: RHSF [{rhsf.pp.f90}{537,3}-{543,16}]
174.081	Loop: RHSF [{rhsf.pp.f90}{545,3}-{551,16}]
124.129	Loop: RHSF [{rhsf.pp.f90}{515,3}-{535,16}]
1153.827	Loop: DERIVATIVE_X_CALC [{derivative_x.pp.f90}{432,10}-{441,15}]
2.177	DERIVATIVE_X_COMM
1422.272	Loop: DERIVATIVE_Y_CALC [{derivative_y.pp.f90}{431,10}-{440,15}]
1459.472	Loop: DERIVATIVE_Z_CALC [{derivative_z.pp.f90}{435,10}-{444,15}]
178.375	Loop: VARIABLES_M::GET_MASS_FRAC [{variables_m.pp.f90}{96,3}-{99,7}]
374.708	Loop: THERMCHEM_M::CALC_INV_AVG_MOL_WT [{thermchem_m.pp.f90}{127,5}-{129,9}]
0	MPI_Irecv()
3.778	DERIVATIVE_Y_COMM
244.308	Loop: THERMCHEM_M::CALC_SPECENTH_ALLPTS [{thermchem_m.pp.f90}{506,3}-{512,8}]
1.294	INTEGRATE
0.002	S3D
7.111	DERIVATIVE_Z_COMM
733.265	Loop: DERIVATIVE_X_CALC [{derivative_x.pp.f90}{490,13}-{509,18}]
765.947	Loop: DERIVATIVE_X_CALC [{derivative_x.pp.f90}{461,13}-{480,18}]
950.372	Loop: DERIVATIVE_Y_CALC [{derivative_y.pp.f90}{488,13}-{507,18}]
973.95	Loop: DERIVATIVE_Y_CALC [{derivative_y.pp.f90}{459,13}-{478,18}]
249.638	Loop: TRANSPORT_M::COMPUTESTRESSTENSOR [{mixavg_transport_m.pp.f90}{711,5}-{737,10}]
1119.025	Loop: DERIVATIVE_Z_CALC [{derivative_z.pp.f90}{464,13}-{483,18}]
0.328	TRANSPORT_M::COMPUTESPECIESDIFFFLUX
1257.338	Loop: DERIVATIVE_Z_CALC [{derivative_z.pp.f90}{493,13}-{512,18}]
84.649	RK_M::CONTROLLER
30.583	DERIVATIVE_X_CALC
33.374	DERIVATIVE_Y_CALC
33.405	DERIVATIVE_Z_CALC
61.032	THERMCHEM_M::CALC_INV_AVG_MOL_WT
95.032	GET_VELOCITY_VEC
103.526	Loop: TRANSPORT_M::COMPUTECOEFFICIENTS [{mixavg_transport_m.pp.f90}{527,5}-{529,10}]
196.839	Loop: FILTER_M::FILTER [{filter_m.pp.f90}{1232,5}-{1236,9}]
293.594	THERMCHEM_M::CALC_TEMP
141.127	TRANSPORT_M::COMPUTECOEFFICIENTS
24.465	COMPUTESCALARGRADIENT
0	Loop: INTEGRATE [{integrate_erk.pp.f90}{108,3}-{110,7}]
0.245	VARIABLES_M::GET_MASS_FRAC

low mflops in loops?

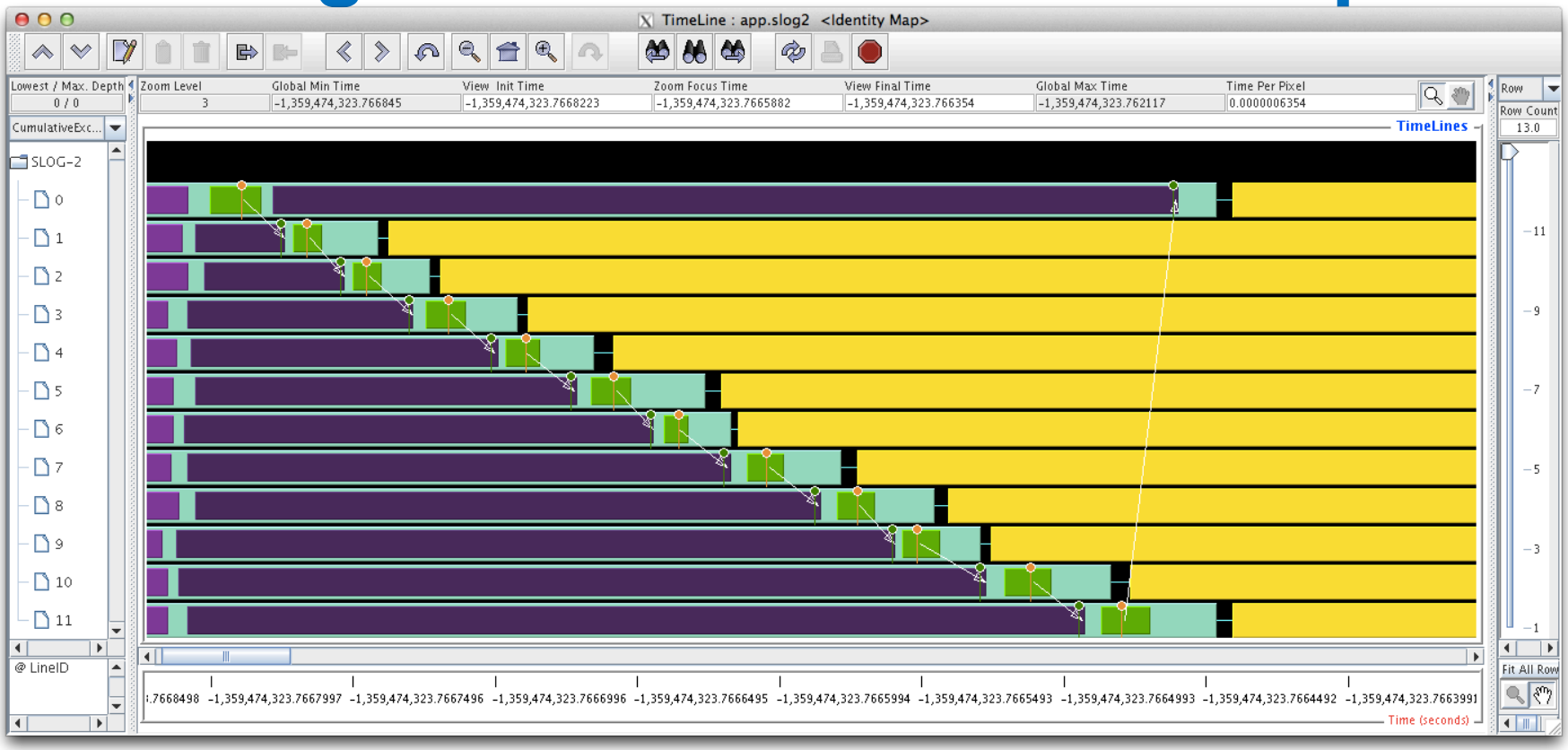
ParaProf 3D Profile Browser



Jumpshot Trace Visualizer in TAU



Tracing Communication in Jumpshot




```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% cmake –DCMAKE_CXX_COMPILER=tau_cxx.sh; make –j 8
% export TAU_TRACE=1
% mpirun –np 16 ./a.out ; tau_treemerge.pl; tau2slog2 tau.trc tau.edf –o a.slog2
% jumpshot a.slog2 &
```

How does TAU work?

Instrumentation: Adds probes to perform measurements

- Source code instrumentation using pre-processors and compiler scripts
- Wrapping external libraries (I/O, MPI, Memory, CUDA, OpenCL, pthread)
- Rewriting the binary executable

Measurement: Profiling or Tracing using wallclock time or hardware counters

- Direct instrumentation (Interval events measure exclusive or inclusive duration)
- Indirect instrumentation (Sampling measures statement level contribution)
- Throttling and runtime control of low-level events that execute frequently
- Per-thread storage of performance data
- Interface with external packages (e.g. PAPI hw performance counter library)

Analysis: Visualization of profiles and traces

- 3D visualization of profile data in paraprof, perfexplorer tools
- Trace conversion & display in external visualizers (Vampir, Jumpshot, ParaVer)

Examples

Using TAU: A Brief Introduction

TAU supports several measurement and thread options

Phase profiling, profiling with hardware counters, MPI library, CUDA…

Each measurement configuration of TAU corresponds to a unique stub makefile and library that is generated when you configure it

To instrument source code automatically using PDT

Choose an appropriate TAU stub makefile in <arch>/lib. On CX1:

% export TAU_MAKEFILE=/phihome/sameer/pkgs/tau_latest/x86_64/lib/Makefile.tau-icpc-papi-mpi-pdt

% export TAU_OPTIONS= '-optVerbose …' (see tau_compiler.sh)

% export PATH=/phihome/sameer/pkgs/tau_latest/x86_64/bin:$PATH
(or source /phihome/sameer/pkgs/tau.x86_64.bashrc/tau.cshrc)

Use tau_f90.sh, tau_cxx.sh, tau_upc.sh, or tau_cc.sh as F90, C++, UPC, or C compilers respectively:

% mpif90 foo.f90 changes to

% tau_f90.sh foo.f90

Set runtime environment variables, execute application and analyze performance data:

% pprof (for text based profile display)

% paraprof (for GUI

Choosing TAU_MAKEFILE

```
% source /phihome/sameer/pkgs/tau.x86_64.bashrc; ls $TAU/Makefile.*
```

Makefile.tau-icpc-papi-mpi-pdt

Makefile.tau-icpc-papi-mpi-python-pdt

Makefile.tau-icpc-papi-ompt-mpi-pdt-openmp

Makefile.tau-papi-mpi-pdt

Makefile.tau-papi-mpi-pdt-openmp

Makefile.tau-papi-mpi-pdt-openmp-opari

Makefile.tau-papi-mpi-python-pdt

For an MPI+F90 application with Intel MPI, you may choose

Makefile.tau-icpc-papi-mpi-pdt

- Supports MPI instrumentation & PDT for automatic source instrumentation

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% tau_f90.sh matrix.f90 -o matrix
% mpirun -np 16 ./matrix
% paraprof
```

Tags in tau_exec and other tools

```
% cd /phihome/sameer/pkgs/tau_latest/x86_64/lib; ls Makefile.*
Makefile.tau-icpc-papi-mpi-pdt

% qsub -I -l select:ncpus=4; mpiexec./matrix
% tau_exec -T icpc,mpi,pdt ./a.out
```

Chooses Makefile.tau-icpc-mpi,pdt and associated libraries.

```
% tau_exec -T serial,pdt ./a.out
```

Chooses Makefile.tau-pdt or the shortest Makefile name without -mpi.

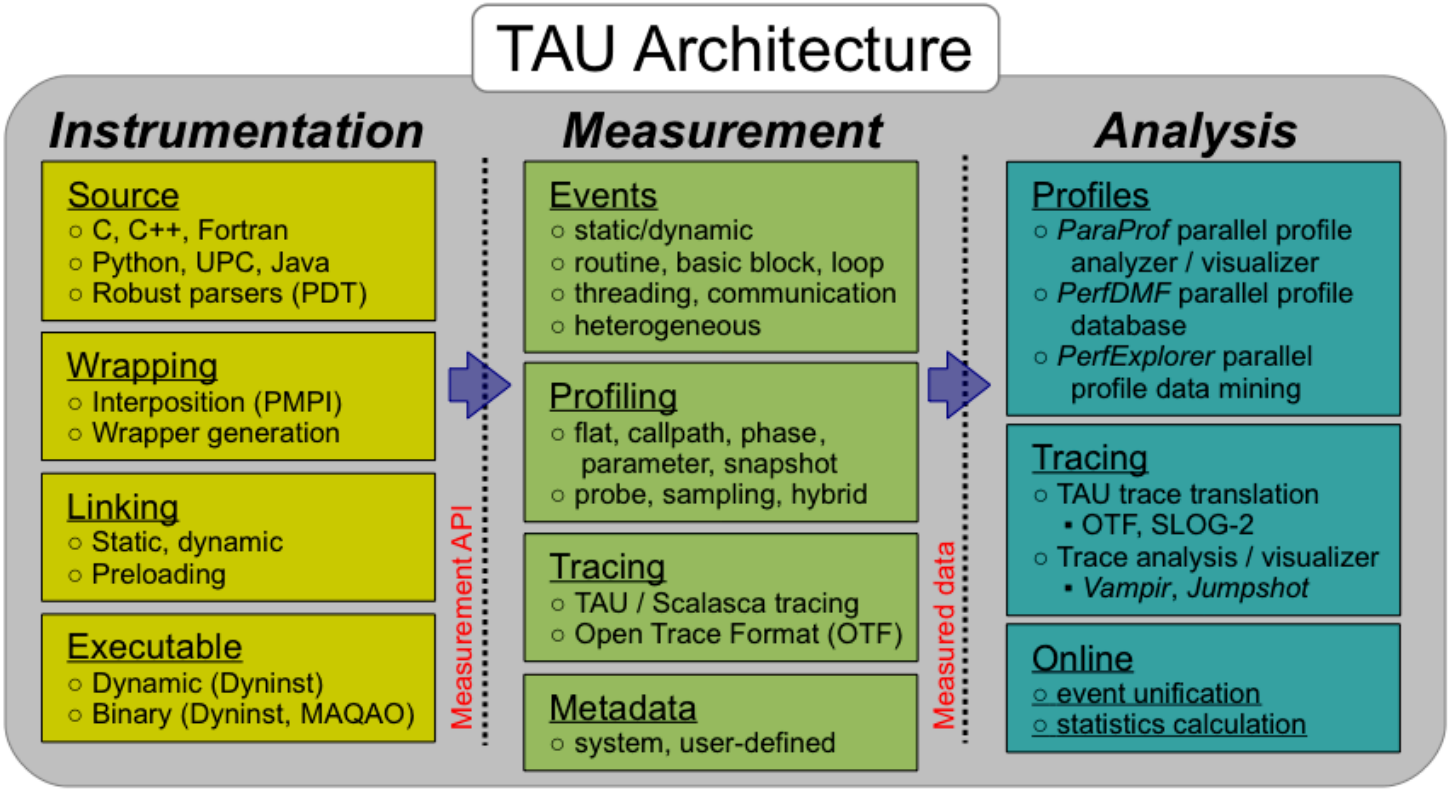
-T <list_of_tags> is used in several TAU tools:

- tau_run
- tau_rewrite
- tau_exec
- tau_gen_wrapper

TAU Performance System®

Parallel performance framework and toolkit

- Goal: to supports all HPC platforms, compilers, and runtime systems
- Provides portable instrumentation, measurement, analysis



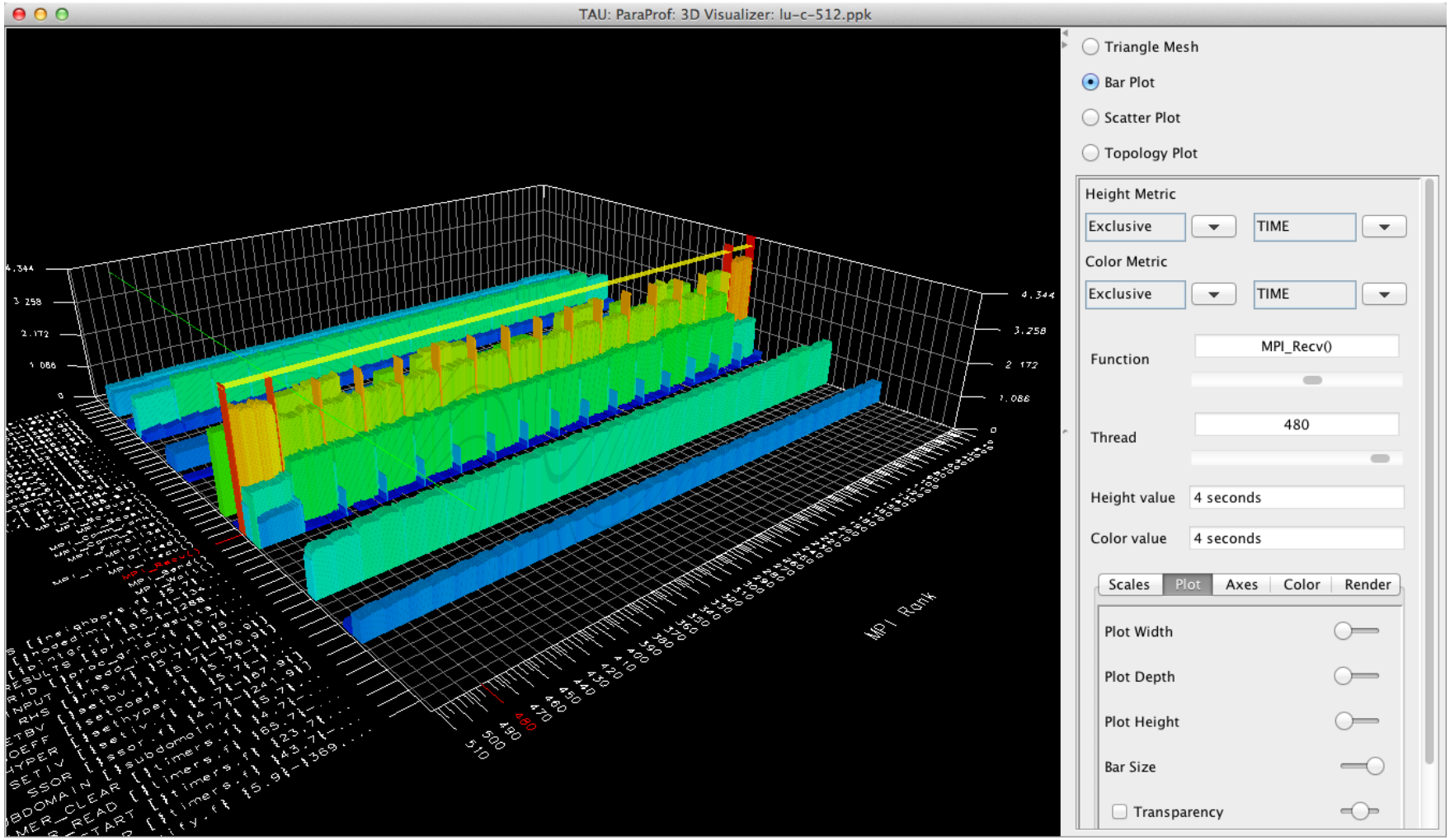
Early Availability on New Systems

- Intel compilers with Intel MPI on Intel Xeon Phi™ (MIC)
- GPI with Intel Linux x86_64 Infiniband clusters
- IBM BG/Q and Power 7 Linux with IBM XL UPC compilers
- NVIDIA Kepler K20 with CUDA 5.0 with NVCC
- Fujitsu Fortran/C/C++ MPI compilers on the K computer
- PGI compilers with OpenACC support on NVIDIA systems
- Cray CX30 Sandybridge Linux systems with Intel compilers
- Cray CCE compilers with OpenACC support on Cray XK7
- AMD OpenCL libs with GNU on AMD Fusion cluster systems
- MPC compilers on TGCC Curie system (Bull, Linux x86_64)
- GNU compilers on ARM Linux clusters (MontBlanc, BSC)
- Cray CCE compilers with OpenACC on Cray XK6 (K20)
- Microsoft MPI with Mingw compilers under Windows Azure
- LLVM and GNU compilers under Mac OS X

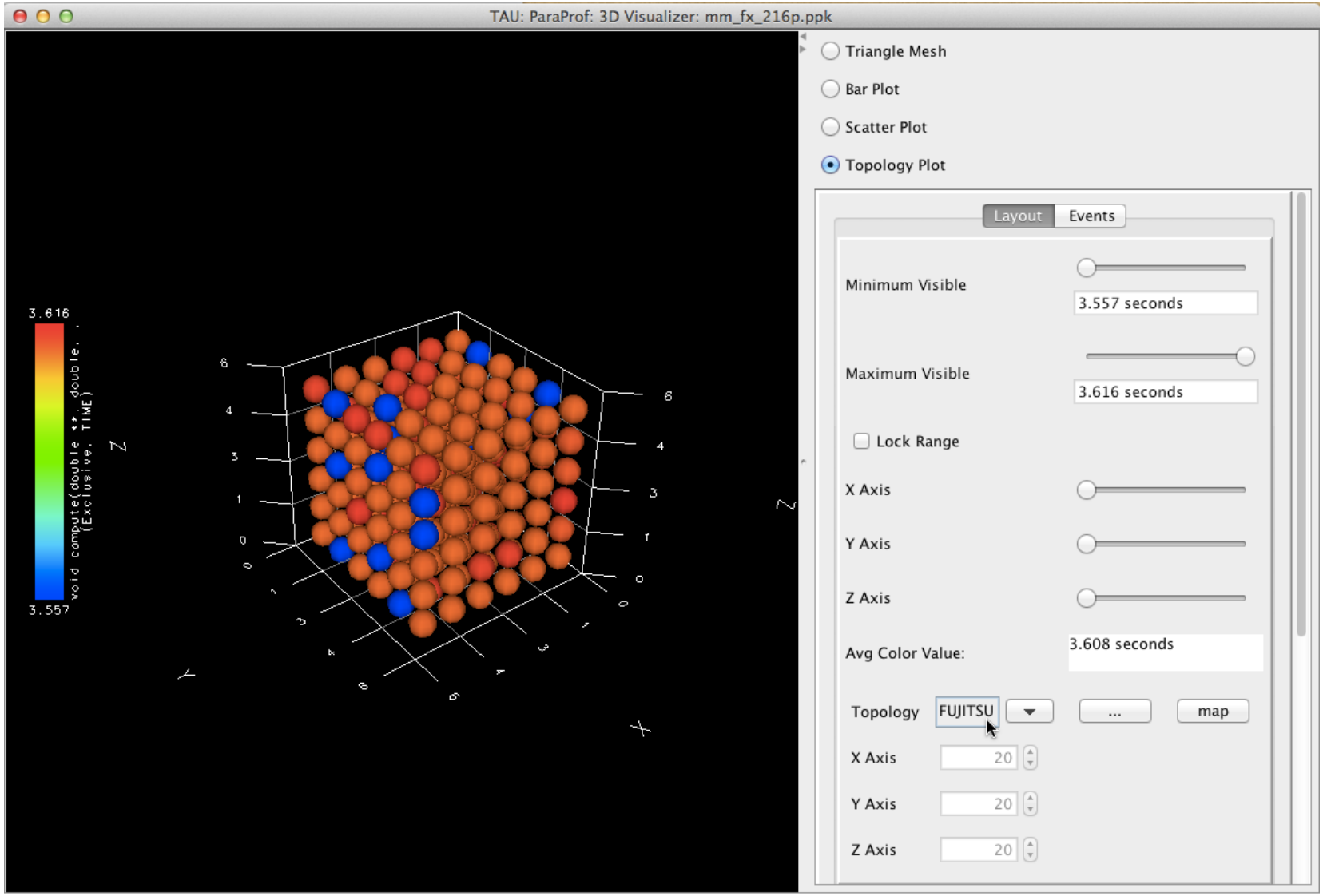
Using TAU on Intel MIC Systems

- **Configuration of PDT:**
  - wget http://tau.uoregon.edu/pdt_lite.tgz
  - ./configure; make ; make install
- **Configuration of TAU for MPI+OpenMP:**
  - wget http://tau.uoregon.edu/tau.tgz
  - ./configure –pdt=<dir> -pdt_c++=g++ -arch=mic_linux -c++=mpiicpc –cc=mpiicc -fortran=mpiifort –mpi –ompt=download –bfd=download
  - make install
- **Compiling:**
  - export TAU_MAKEFILE=<taudir>/mic_linux/ lib/Makefile.tau-icpc-ompt-mpi-pdt-openmp
  - make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh
  - export LD_LIBRARY_PATH=<taudir>/mic_linux/lib:$LD_LIBRARY_PATH

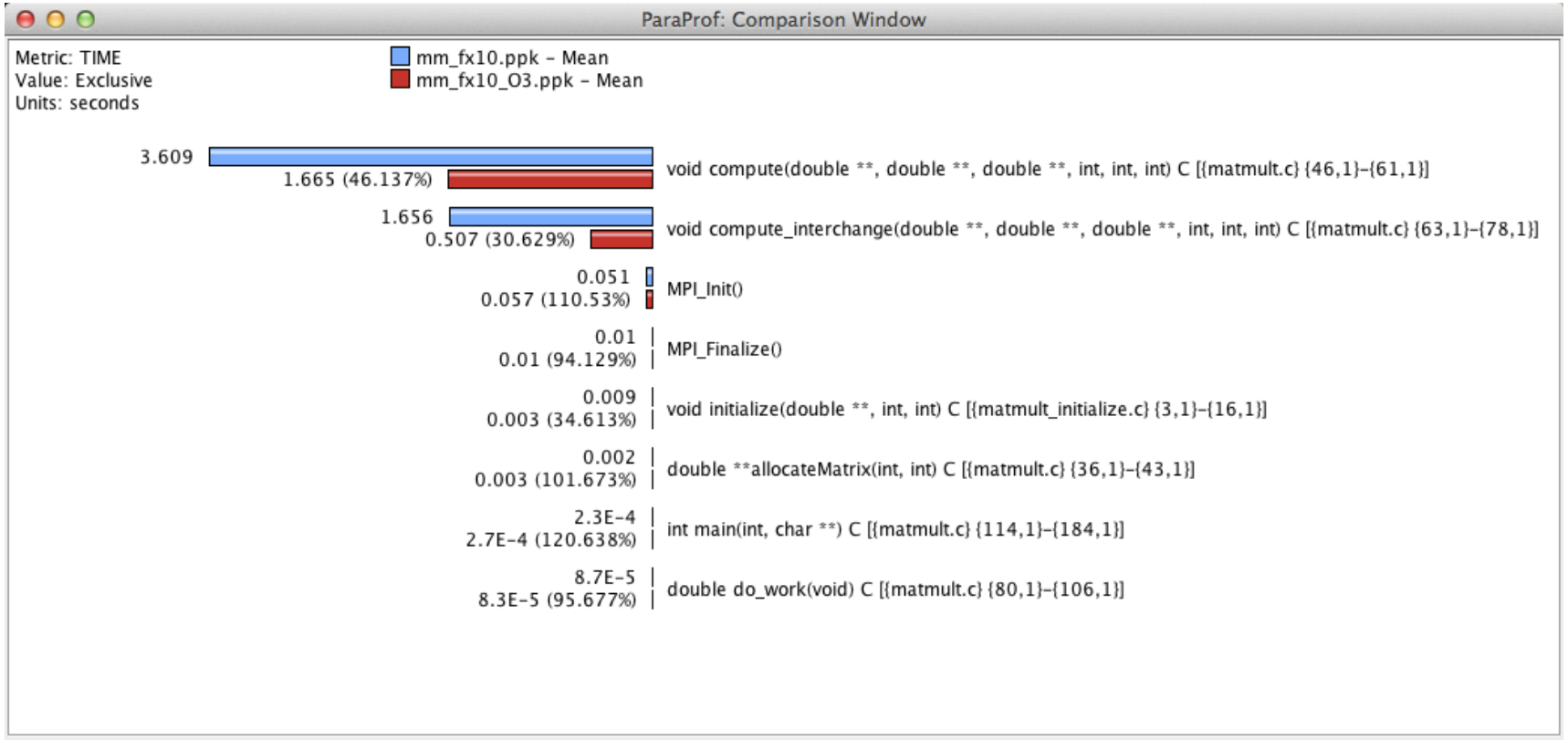
TAU's ParaProf 3D Browser



TAU's ParaProf Topology Displays



TAU's ParaProf Comparison Window



Event Based Sampling in TAU

TAU: ParaProf: Statistics for: node 0 - /usr/global/tools/tau/training/tau-2.23/examples/mm

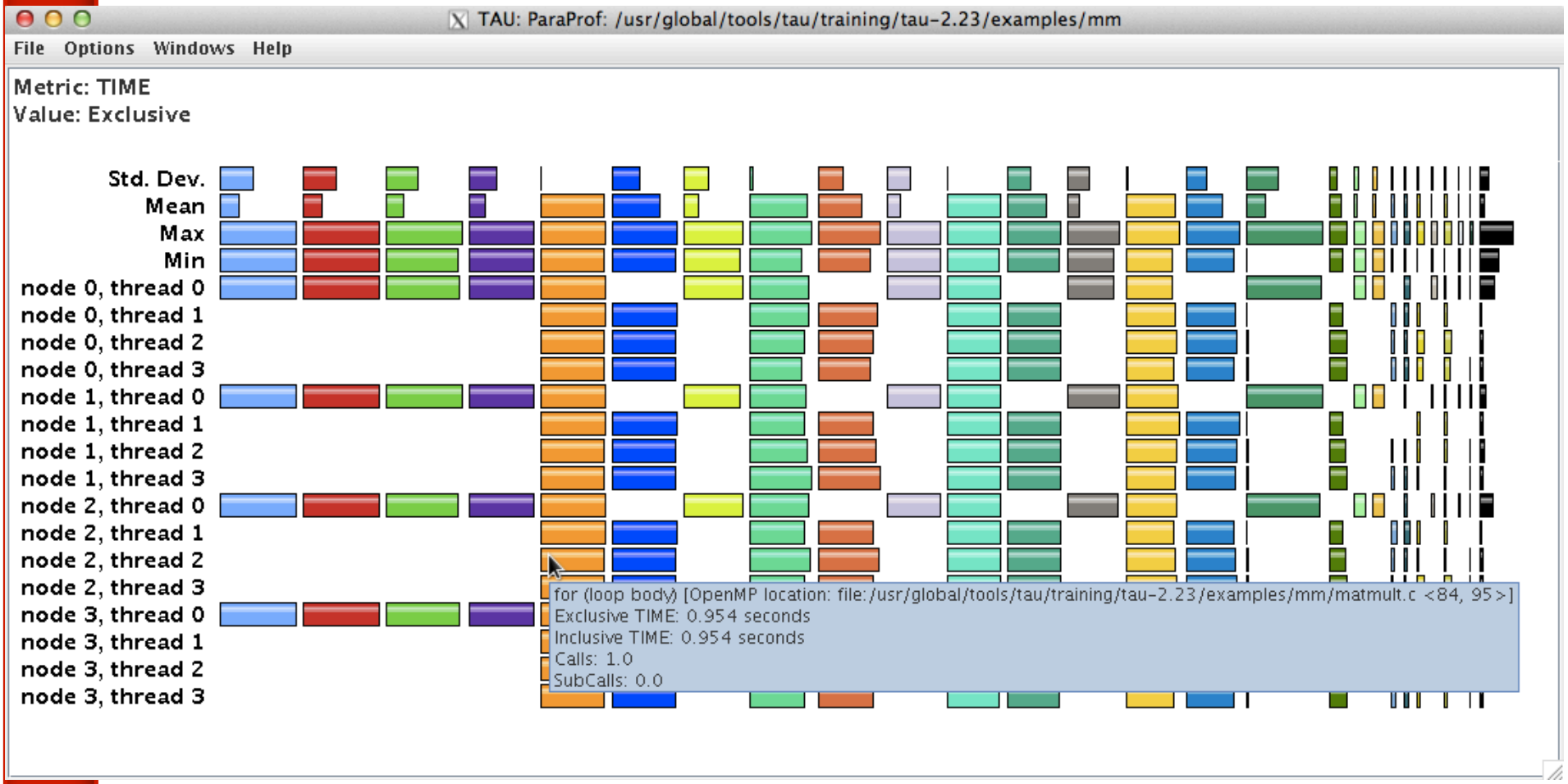
File Options Windows Help

Name	Exclusive TIME	Inclusive TIME	Calls	Child Calls
int main(int, char **) C [{matmult.c} {159,1}-{229,1}]	0.033	2,837.969	1	3
MPI_Finalize()	16.403	16.444	1	5
MPI_Init()	1,140.687	1,141.011	1	45
double do_work(void) C [{matmult.c} {120,1}-{151,1}]	0.041	1,680.481	1	8
double **allocateMatrix(int, int) C [{matmult.c} {36,1}-{43,1}]	2.959	2.959	3	0
void compute(double **, double **, double **, int, int, int) C [{matmult.c} {78,1}-{97,1}]	956.539	956.539	1	0
[CONTEXT] void compute(double **, double **, double **, int, int, int) C [{matmult.c} {78,1}-{97,1}]	0	949.969	95	0
[SAMPLE] compute [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c} {87}]	59.993	59.993	6	0
[SAMPLE] compute [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c} {91}]	889.976	889.976	89	0
void compute_interchange(double **, double **, double **, int, int, int) C [{matmult.c} {99,1}-{118,1}]	718.179	718.179	1	0
[CONTEXT] void compute_interchange(double **, double **, double **, int, int, int) C [{matmult.c} {99,1}-{118,1}]	0	720	72	0
[SAMPLE] compute_interchange [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c} {108}]	60	60	6	0
[SAMPLE] compute_interchange [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c} {112}]	660	660	66	0
void initialize(double **, int, int) C [{matmult_initialize.c} {3,1}-{16,1}]	2.763	2.763	3	0

Show Source Code
Show Function Bar Chart
Show Function Histogram
Assign Function Color
Reset to Default Color

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% make CC=tau_cc.sh CXX=tau_cxx.sh
% export TAU_SAMPLING=1
% mpirun –np 256 ./a.out
% paraprof
```

Mixed MPI and OpenMP Instrumentation



Opari OpenMP Instrumentation, Sampling

TAU: ParaProf: Statistics for: node 0, thread 0 - /usr/global/tools/tau/training/tau-2.23/examples/mm

File Options Windows Help

Name	Exclu...	Inclu...	C...	C...
int main(int, char **) C [{matmult.pomp.c} {224,1}-{294,1}]	0	3.134	1	3
MPI_Finalize()	0.191	0.191	1	5
MPI_Init_thread()	1.148	1.151	1	45
double do_work(void) C [{matmult.pomp.c} {185,1}-{216,1}]	0.001	1.792	1	8
double **allocateMatrix(int, int) C [{matmult.pomp.c} {38,1}-{45,1}]	0.003	0.003	3	0
void compute(double **, double **, double **, int, int, int) C [{matmult.pomp.c} {111,1}-{146,1}]	0	0.97	1	1
parallel fork/join [OpenMP]	0	0.97	1	1
parallel (parallel fork/join) [OpenMP location: file:/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c <80, 96>]	0	0.97	1	1
parallel begin/end [OpenMP]	0	0.97	1	1
parallel (parallel begin/end) [OpenMP location: file:/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c <80, 96>]	0	0.97	1	2
barrier enter/exit [OpenMP]	0	0.003	1	1
parallel (barrier enter/exit) [OpenMP location: file:/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c <80, 96>]	0.003	0.003	1	0
for enter/exit [OpenMP]	0	0.967	1	1
for (loop body) [OpenMP location: file:/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c <84, 95>]	0.967	0.967	1	0
[CONTEXT] for (loop body) [OpenMP location: file:/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c <84, 95>]	0	0.93	58	0
[SAMPLE] L_compute_118__par_region0_2_204 [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.pomp.c} {127}]	0.039	0.039	3	0
[SAMPLE] L_compute_118__par_region0_2_204 [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.pomp.c} {131}]	0.891	0.891	55	0
void compute_interchange(double **, double **, double **, int, int, int) C [{matmult.pomp.c} {148,1}-{183,1}]	0	0.812	1	1
parallel fork/join [OpenMP]	0	0.812	1	1
parallel (parallel fork/join) [OpenMP location: file:/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c <101, 117>]	0	0.811	1	1
parallel begin/end [OpenMP]	0	0.811	1	1
parallel (parallel begin/end) [OpenMP location: file:/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c <101, 117>]	0	0.811	1	2
void initialize(double **, int, int) C [{matmult_initialize.pomp.c} {5,1}-{33,1}]	0	0.007	3	3

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt-opari-openmp
% make CC=tau_cc.sh CXX=tau_cxx.sh
% export TAU_SAMPLING=1; export OMP_NUM_THREADS=16
% mpirun –np 256 ./a.out
% paraprof
```

TAU's support for Intel OMPT

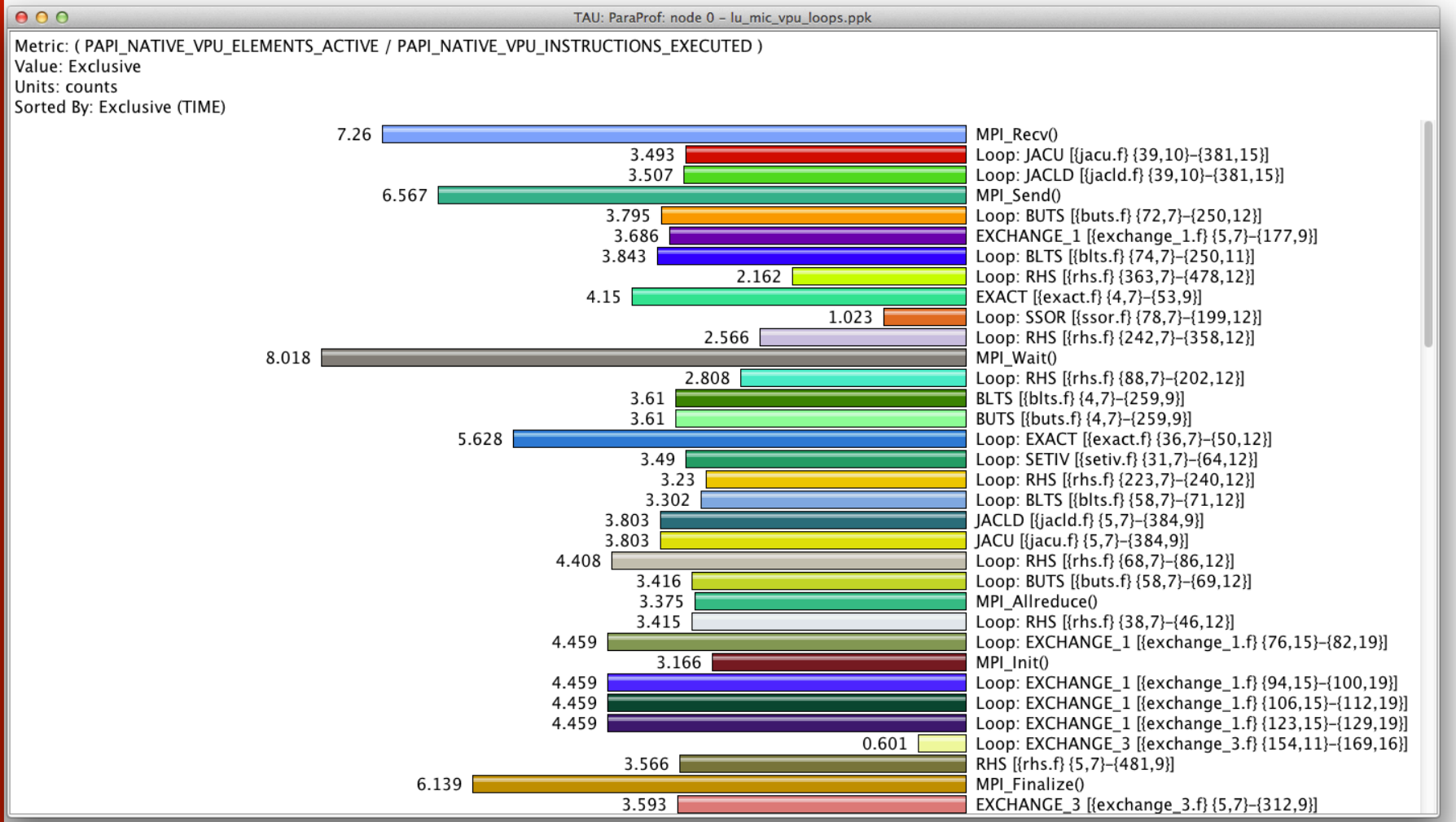
TAU: ParaProf: Statistics for: node 0, thread 0 – /usr/global/tools/tau/training/tau-2.23/examples/mm

File Options Windows Help

Name	Exclusi...	Inclusi...	Calls	Child Calls
.TAU application	0	2.943	1	1
int main(int, char **) C [{matmult.c} {159,1}-{229,1}]	0	2.942	1	3
MPI_Finalize()	0.034	0.034	1	5
MPI_Init_thread()	1.148	1.151	1	45
double do_work(void) C [{matmult.c} {120,1}-{151,1}]	0	1.757	1	8
double **allocateMatrix(int, int) C [{matmult.c} {36,1}-{43,1}]	0.003	0.003	3	0
void compute(double **, double **, double **, int, int, int) C [{matmult.c} {78,1}-{97,1}]	0	0.951	1	1
OpenMP_PARALLEL_REGION: [OPENMP] compute [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c} {80}]	0	0.951	1	2
OpenMP_BARRIER: [OPENMP] compute [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c} {80}]	0.004	0.004	1	0
[CONTEXT] OpenMP_BARRIER: [OPENMP] compute [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c} {80}]	0	0.007	1	0
OpenMP_LOOP: [OPENMP] compute [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c} {80}]	0.946	0.946	1	0
[CONTEXT] OpenMP_LOOP: [OPENMP] compute [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c} {80}]	0	0.944	62	0
[SAMPLE] L_compute_80__par_region0_2_150 [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c} {87}]	0.047	0.047	2	0
[SAMPLE] L_compute_80__par_region0_2_150 [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c} {91}]	0.897	0.897	60	0
void compute_interchange(double **, double **, double **, int, int, int) C [{matmult.c} {99,1}-{118,1}]	0	0.787	1	1
void initialize(double **, int, int) C [{matmult_initialize.c} {3,1}-{16,1}]	0.005	0.016	3	3
OpenMP_PARALLEL_REGION: [OPENMP] initialize [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult_initialize.c} {5}	0	0.011	3	6
OpenMP_BARRIER: [OPENMP] initialize [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult_initialize.c} {5}]	0.01	0.01	3	0
[CONTEXT] OpenMP_BARRIER: [OPENMP] initialize [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult_initializ	0	0.03	2	0
OpenMP_LOOP: [OPENMP] initialize [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult_initialize.c} {5}]	0.001	0.001	3	0

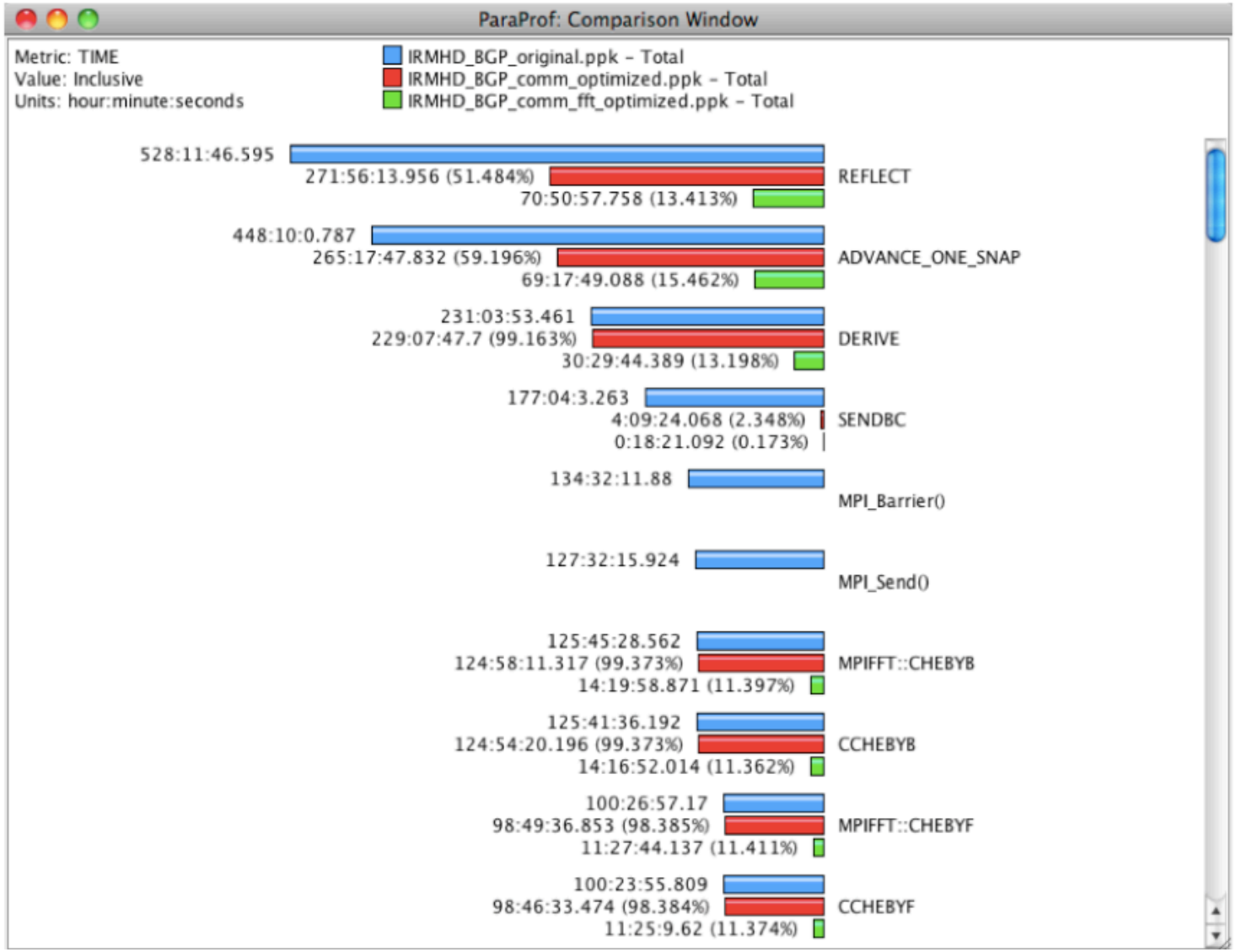
```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt-ompt-openmp
% make CC=tau_cc.sh CXX=tau_cxx.sh
% export TAU_SAMPLING=1; export OMP_NUM_THREADS=16
% mpirun –np 256 tau_exec –T ompt –loadlib=$TAU/libiomp5.so  ./a.out
% paraprof
```

ParaProf's Derived Metric Window: MIC

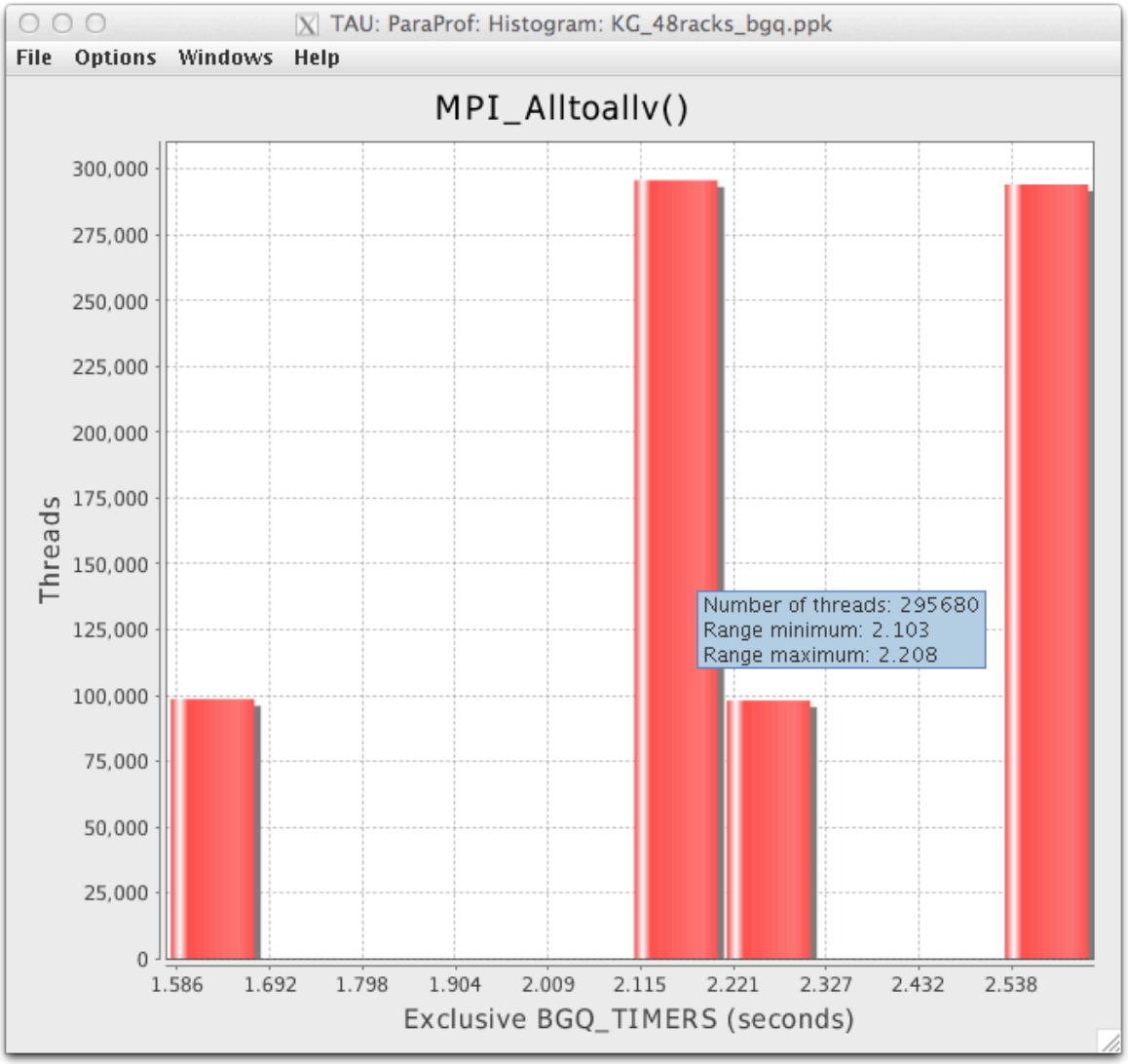



```
% export TAU_MAKEFILE=$TAUROOT/mic_linux/lib/Makefile.tau-icpc-papi-mpi-pdt
% export TAU_METRICS=TIME,PAPI_NATIVE_VPU_ELEMENTS_ACTIVE…
```

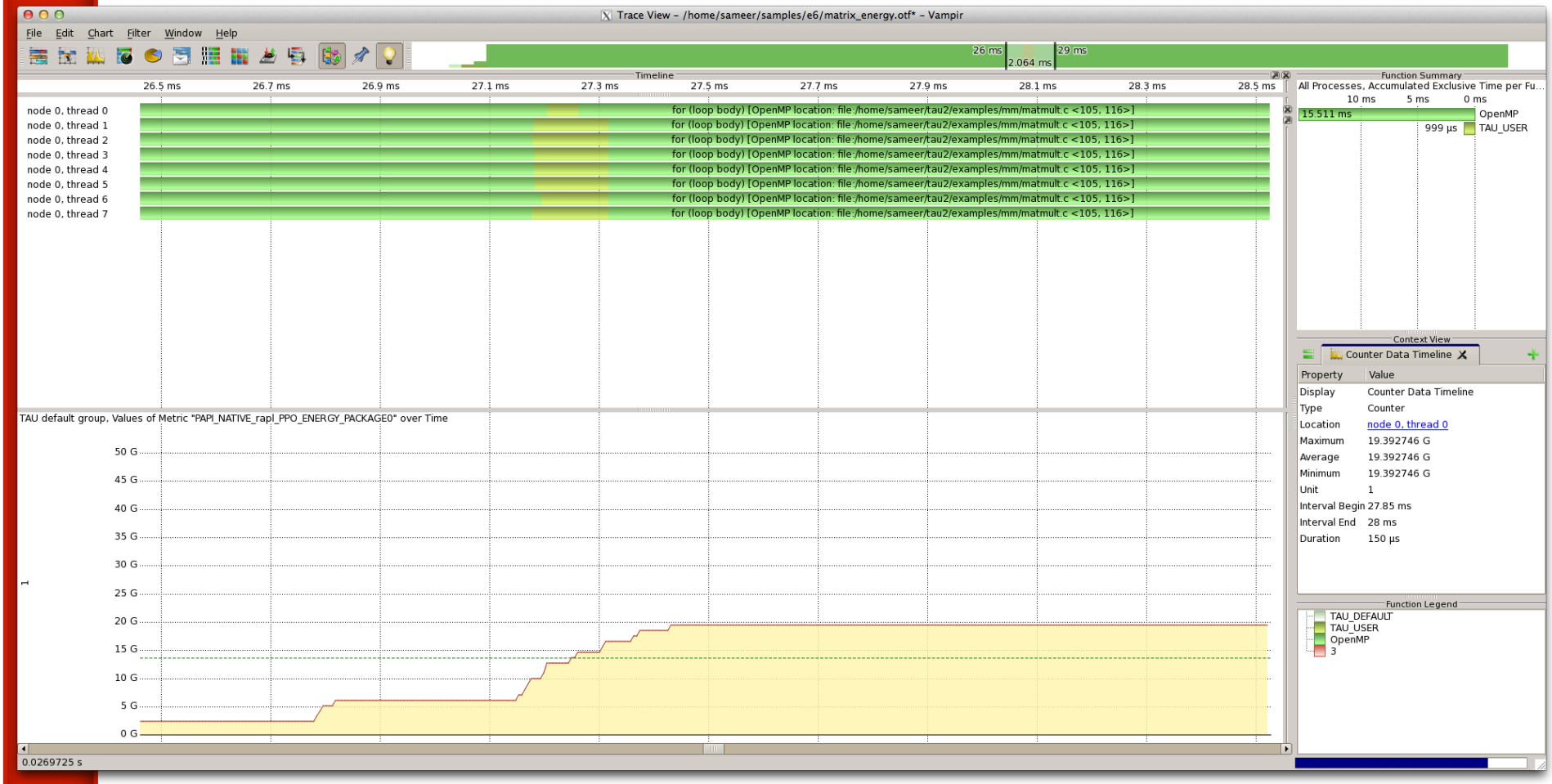
ParaProf Comparison Window



ParaProf Histogram Display



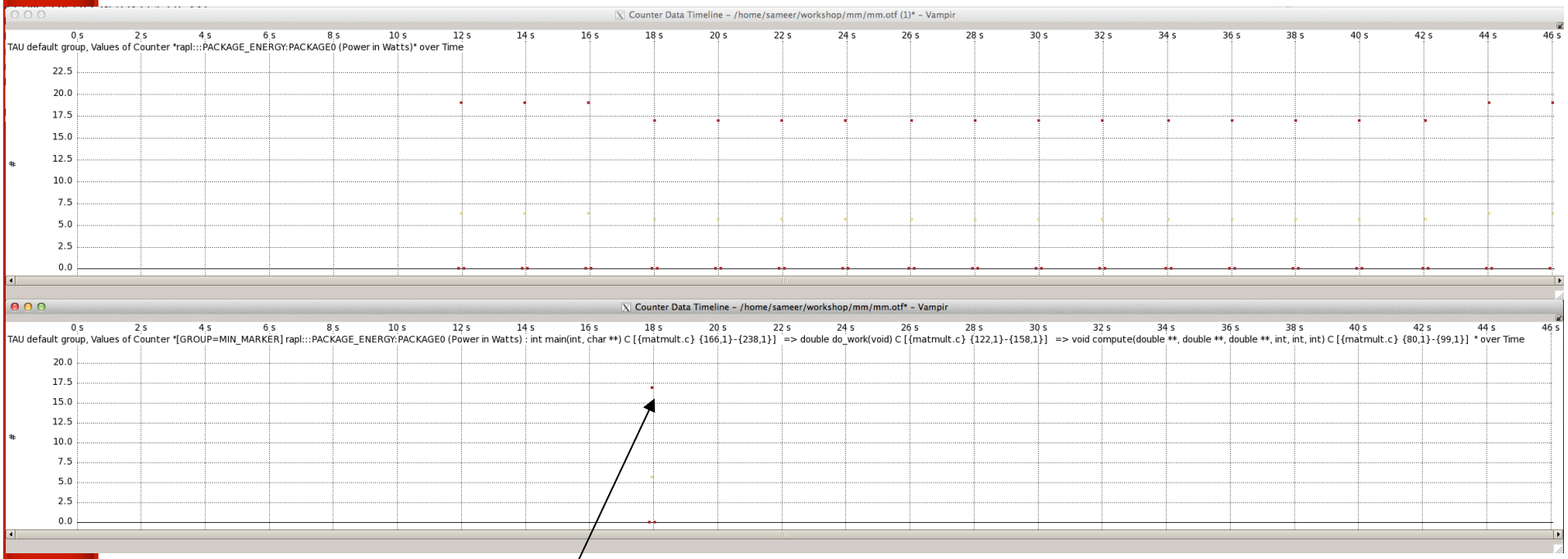
Tracing Energy Usage with TAU and Vampir



ParaTools

UNIVERSITY OF OREGON

Marker Events: Tracing



When an atomic event exceeds the max or min value by a threshold (say 20%), a marker context event is triggered to record the callstack.

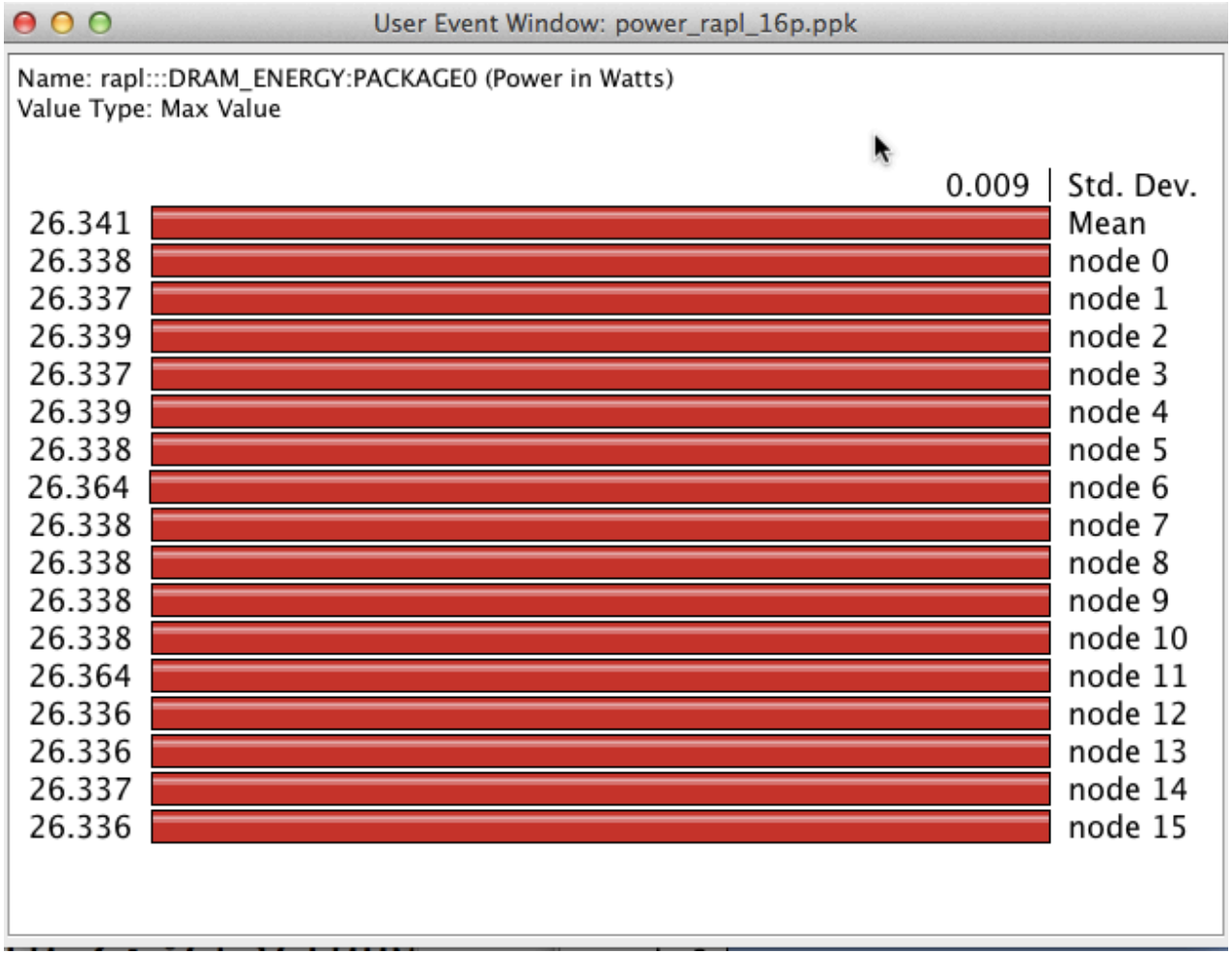
Marker events show sudden spikes

TAU: ParaProf: Context Events for: node 2 – rapl_marker_16p.ppk

Name	MaxValue	MinValue	NumSamples	MeanValue	Std. Dev.
int main(int, char **) C [{matmult.c}{165,1}-{237,1}]					
double do_work(void) C [{matmult.c}{126,1}-{157,1}]					
void compute(double **, double **, double **, int, int, int) C [{matmult.c}{84,1}-{103,1}]					
[GROUP=MAX_MARKER] rapl:::DRAM_ENERGY:PACKAGE0 (Power in Watts)	17.585	17.469	5	17.521	0.037
[GROUP=MAX_MARKER] rapl:::DRAM_ENERGY:PACKAGE1 (Power in Watts)	15.261	15.218	4	15.237	0.016
[GROUP=MAX_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE0 (Power in Watts)	118.903	114.923	22	116.98	1.201
[GROUP=MAX_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE1 (Power in Watts)	113.466	110.207	22	111.778	0.996
[GROUP=MAX_MARKER] rapl:::PP0_ENERGY:PACKAGE0 (Power in Watts)	100.138	96.266	24	98.206	1.13
[GROUP=MAX_MARKER] rapl:::PP0_ENERGY:PACKAGE1 (Power in Watts)	95.846	92.758	24	94.319	0.937
[GROUP=MIN_MARKER] rapl:::DRAM_ENERGY:PACKAGE0 (Power in Watts)	17.397	17.303	4	17.358	0.035
[GROUP=MIN_MARKER] rapl:::DRAM_ENERGY:PACKAGE1 (Power in Watts)	15.048	15.042	2	15.045	0.003
int mysleep(int) C [{matmult.c}{46,1}-{49,1}]					
[GROUP=MIN_MARKER] rapl:::DRAM_ENERGY:PACKAGE0 (Power in Watts)	15.84	15.84	1	15.84	0
[GROUP=MIN_MARKER] rapl:::DRAM_ENERGY:PACKAGE1 (Power in Watts)	14.275	14.275	1	14.275	0
[GROUP=MIN_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE1 (Power in Watts)	96.853	96.853	1	96.853	0
[GROUP=MIN_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE0 (Power in Watts)	93.125	93.125	1	93.125	0
[GROUP=MIN_MARKER] rapl:::PP0_ENERGY:PACKAGE0 (Power in Watts)	75.096	75.096	1	75.096	0
[GROUP=MIN_MARKER] rapl:::PP0_ENERGY:PACKAGE1 (Power in Watts)	79.646	79.646	1	79.646	0
void compute_interchange(double **, double **, double **, int, int, int) C [{matmult.c}{105,1}-{124,1}]					
[GROUP=MAX_MARKER] rapl:::DRAM_ENERGY:PACKAGE0 (Power in Watts)	26.064	25.711	2	25.887	0.176
[GROUP=MAX_MARKER] rapl:::DRAM_ENERGY:PACKAGE1 (Power in Watts)	24.373	23.965	4	24.232	0.159
[GROUP=MAX_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE0 (Power in Watts)	126.872	125.182	6	125.732	0.557
[GROUP=MAX_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE1 (Power in Watts)	124.377	116.689	5	122.428	2.885
[GROUP=MAX_MARKER] rapl:::PP0_ENERGY:PACKAGE0 (Power in Watts)	103.981	102.21	6	102.769	0.584
[GROUP=MAX_MARKER] rapl:::PP0_ENERGY:PACKAGE1 (Power in Watts)	102.615	101.693	4	102.115	0.33
rapl:::DRAM_ENERGY:PACKAGE0 (Power in Watts)	26.064	15.84	36	19.053	3.39
rapl:::DRAM_ENERGY:PACKAGE1 (Power in Watts)	24.373	14.275	36	16.435	3.155
rapl:::PACKAGE_ENERGY:PACKAGE0 (Power in Watts)	126.872	93.125	36	117.729	5.403
rapl:::PACKAGE_ENERGY:PACKAGE1 (Power in Watts)	124.377	96.853	36	112.961	4.776
rapl:::PP0_ENERGY:PACKAGE0 (Power in Watts)	103.981	75.096	36	98.208	4.466
rapl:::PP0_ENERGY:PACKAGE1 (Power in Watts)	102.615	79.646	36	94.872	3.662

```
% export TAU_EVENT_THRESHOLD 0.5
```

Profiling in TAU



Power Profiling using PAPI+RAPL

TAU: ParaProf: Context Events for: node 0 – power_rapl_16p.ppk

Name	..	NumSamples	MaxValue	MinValue	MeanValue	Std. Dev.
rapl:::DRAM_ENERGY:PACKAGE0 (Power in Watts)	...	39	26.338	17.673	20.514	3.031
rapl:::DRAM_ENERGY:PACKAGE1 (Power in Watts)	...	39	21.29	11.462	14.182	2.275
rapl:::PACKAGE_ENERGY:PACKAGE0 (Power in Watts)	...	39	120.72	80.665	114.013	7.222
rapl:::PACKAGE_ENERGY:PACKAGE1 (Power in Watts)	...	39	117.461	77.758	106.762	6.365
rapl:::PP0_ENERGY:PACKAGE0 (Power in Watts)	...	39	97.753	61.266	93.768	7.06
rapl:::PP0_ENERGY:PACKAGE1 (Power in Watts)	...	39	97.987	61.895	89.887	6.12

```
#include <TAU.h>

TAU_TRACK_POWER();   // In Fortran: call TAU_TRACK_POWER()

% sudo chmod –R go+r /dev/cpu/*/msr
% sudo /sbin/setcap cap_sys_rawio=ep ./a.out
% unset LD_LIBRARY_PATH
% ldd ./a.out
should have no “not found” entries, Use –Wl,-rpath,/path while linking
% ./a.out
% paraprof
```

TAU Instrumentation Approach

Supports both direct and indirect performance observation

- Direct instrumentation of program (system) code (probes)
- Instrumentation invokes performance measurement
- Event measurement: performance data, meta-data, context
- Indirect mode supports sampling based on periodic timer or hardware performance counter overflow based interrupts

Support for user-defined events

- ***Interval*** (Start/Stop) events to measure exclusive & inclusive duration
- ***Atomic events*** (Trigger at a single point with data, e.g., heap memory)
  - Measures total, samples, min/max/mean/std. deviation statistics
- ***Context events*** (are atomic events with executing context)
  - Measures above statistics for a given calling path

Direct Observation: Events

Event types

- Interval events (begin/end events)
  - Measures exclusive & inclusive durations between events
  - Metrics monotonically increase
- Atomic events (trigger with data value)
  - Used to capture performance data state
  - Shows extent of variation of triggered values (min/max/mean)

Code events

- Routines, classes, templates
- Statement-level blocks, loops

Interval and Atomic Events in TAU

```
xterm
NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.187        1,105           1          44    1105659 int main(int, char **) C
 93.2        1,030        1,030           1           0    1030654 MPI_Init()
  5.9        0.879           65          40         320       1637 void func(int, int) C
  4.6           51           51          40           0       1277 MPI_Barrier()
  1.2           13           13         120           0        111 MPI_Recv()
  0.8            9            9           1           0       9328 MPI_Finalize()
  0.0        0.137        0.137         120           0          1 MPI_Send()
  0.0        0.086        0.086          40           0          2 MPI_Bcast()
  0.0        0.002        0.002           1           0          2 MPI_Comm_size()
  0.0        0.001        0.001           1           0          1 MPI_Comm_rank()
---------------------------------------------------------------------------------------

USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
---------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue  MeanValue  Std. Dev.  Event Name
---------------------------------------------------------------------------------------
       365  5.138E+04      44.39   3.09E+04  1.234E+04  Heap Memory Used (KB) : Entry
       365  5.138E+04       2064  3.115E+04   1.21E+04  Heap Memory Used (KB) : Exit
        40         40         40         40          0  Message size for broadcast
---------------------------------------------------------------------------------------
                                                                                 27,1
```

Interval events e.g., routines (start/stop) show ***duration***

Atomic events (triggered with value) show ***extent of variation*** (min/max/mean)

% export TAU_CALLPATH_DEPTH=0
% export TAU_TRACK_HEAP=1

Atomic Events, Context Events

```
xterm
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.253        1,106           1          44    1106701 int main(int, char **) C
 93.2        1,031        1,031           1           0    1031311 MPI_Init()
  6.0            1           66          40         320       1650 void func(int, int) C
  5.7           63           63          40           0       1588 MPI_Barrier()
  0.8            9            9           1           0       9119 MPI_Finalize()
  0.1            1            1         120           0         10 MPI_Recv()
  0.0        0.141        0.141         120           0          1 MPI_Send()
  0.0        0.085        0.085          40           0          2 MPI_Bcast()
  0.0        0.001        0.001           1           0          1 MPI_Comm_size()
  0.0            0            0           1           0          0 MPI_Comm_rank()
---------------------------------------------------------------------------------------

USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
---------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue  MeanValue  Std. Dev.  Event Name
---------------------------------------------------------------------------------------
        40         40         40         40          0  Message size for broadcast
       365  5.139E+04      44.39  3.091E+04  1.234E+04  Heap Memory Used (KB) : Entry
        40  5.139E+04       3097  3.114E+04  1.227E+04  Heap Memory Used (KB) : Entry : MPI_Barrier()
        40  5.139E+04   1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : MPI_Bcast()
         1       2067       2067       2067          0  Heap Memory Used (KB) : Entry : MPI_Comm_rank()
         1       2066       2066       2066          0  Heap Memory Used (KB) : Entry : MPI_Comm_size()
         1  5.139E+04  5.139E+04  5.139E+04  0.0006905  Heap Memory Used (KB) : Entry : MPI_Finalize()
         1      57.56      57.56      57.56          0  Heap Memory Used (KB) : Entry : MPI_Init()
       120  5.139E+04   1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : MPI_Recv()
       120  5.139E+04  1.129E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : MPI_Send()
         1      44.39      44.39      44.39          0  Heap Memory Used (KB) : Entry : int main(int, char **) C
        40  5.036E+04       2068  3.011E+04  1.227E+04  Heap Memory Used (KB) : Entry : void func(int, int) C
                                                                                   4.9          1%
```

Atomic events

Context events =atomic event + executing context

% export TAU_CALLPATH_DEPTH=1

Controls depth of executing context shown in profiles

% export TAU_TRACK_HEAP=1

Context Events (Default)

```
xterm
NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.357        1,114           1          44    1114040 int main(int, char **) C
 92.6        1,031        1,031           1           0    1031066 MPI_Init()
  6.7           72           74          40         320       1865 void func(int, int) C
  0.7            8            8           1           0       8002 MPI_Finalize()
  0.1            1            1         120           0         12 MPI_Recv()
  0.1        0.608        0.608          40           0         15 MPI_Barrier()
  0.0        0.136        0.136         120           0          1 MPI_Send()
  0.0        0.095        0.095          40           0          2 MPI_Bcast()
  0.0        0.001        0.001           1           0          1 MPI_Comm_size()
  0.0            0            0           1           0          0 MPI_Comm_rank()
---------------------------------------------------------------------------------------

USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
---------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue  MeanValue  Std. Dev.  Event Name
---------------------------------------------------------------------------------------
       365  5.139E+04      44.39  3.091E+04  1.234E+04  Heap Memory Used (KB) : Entry
         1      44.39      44.39      44.39          0  Heap Memory Used (KB) : Entry : int main(int, char **) C
         1       2068       2068       2068          0  Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Comm_rank()
         1       2066       2066       2066          0  Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Comm_size()
         1  5.139E+04  5.139E+04  5.139E+04          0  Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Finalize()
         1      57.58      57.58      57.58          0  Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Init()
        40  5.036E+04       2069  3.011E+04  1.228E+04  Heap Memory Used (KB) : Entry : int main(int, char **) C => void func(int, int) C
        40  5.139E+04       3098  3.114E+04  1.227E+04  Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Barrier()
        40  5.139E+04    1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Bcast()
       120  5.139E+04    1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Recv()
       120  5.139E+04    1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Send()
       365  5.139E+04       2065  3.116E+04   1.21E+04  Heap Memory Used (KB) : Exit
```

Context event =atomic event + executing context

% export TAU_CALLPATH_DEPTH=2
% export TAU_TRACK_HEAP=1

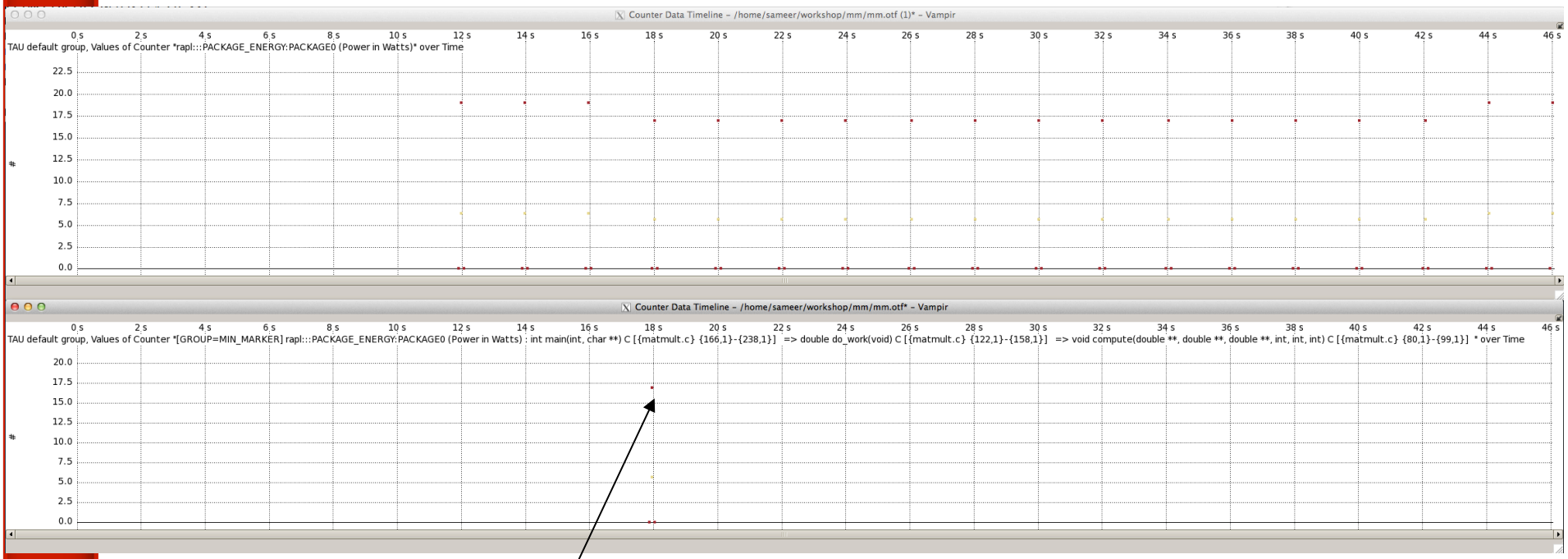
Marker Events: Profiling

TAU: ParaProf: Context Events for: node 2 - rapl_marker_16p.ppk

Name	MaxValue	MinValue	NumSamples	MeanValue	Std. Dev.
int main(int, char **) C [{matmult.c}{165,1}-{237,1}]					
double do_work(void) C [{matmult.c}{126,1}-{157,1}]					
void compute(double **, double **, double **, int, int, int) C [{matmult.c}{84,1}-{103,1}]					
[GROUP=MAX_MARKER] rapl:::DRAM_ENERGY:PACKAGE0 (Power in Watts)	17.585	17.469	5	17.521	0.037
[GROUP=MAX_MARKER] rapl:::DRAM_ENERGY:PACKAGE1 (Power in Watts)	15.261	15.218	4	15.237	0.016
[GROUP=MAX_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE0 (Power in Watts)	118.903	114.923	22	116.98	1.201
[GROUP=MAX_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE1 (Power in Watts)	113.466	110.207	22	111.778	0.996
[GROUP=MAX_MARKER] rapl:::PP0_ENERGY:PACKAGE0 (Power in Watts)	100.138	96.266	24	98.206	1.13
[GROUP=MAX_MARKER] rapl:::PP0_ENERGY:PACKAGE1 (Power in Watts)	95.846	92.758	24	94.319	0.937
[GROUP=MIN_MARKER] rapl:::DRAM_ENERGY:PACKAGE0 (Power in Watts)	17.397	17.303	4	17.358	0.035
[GROUP=MIN_MARKER] rapl:::DRAM_ENERGY:PACKAGE1 (Power in Watts)	15.048	15.042	2	15.045	0.003
int mysleep(int) C [{matmult.c}{46,1}-{49,1}]					
[GROUP=MIN_MARKER] rapl:::DRAM_ENERGY:PACKAGE0 (Power in Watts)	15.84	15.84	1	15.84	0
[GROUP=MIN_MARKER] rapl:::DRAM_ENERGY:PACKAGE1 (Power in Watts)	14.275	14.275	1	14.275	0
[GROUP=MIN_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE1 (Power in Watts)	96.853	96.853	1	96.853	0
[GROUP=MIN_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE0 (Power in Watts)	93.125	93.125	1	93.125	0
[GROUP=MIN_MARKER] rapl:::PP0_ENERGY:PACKAGE0 (Power in Watts)	75.096	75.096	1	75.096	0
[GROUP=MIN_MARKER] rapl:::PP0_ENERGY:PACKAGE1 (Power in Watts)	79.646	79.646	1	79.646	0
void compute_interchange(double **, double **, double **, int, int, int) C [{matmult.c}{105,1}-{124,1}]					
[GROUP=MAX_MARKER] rapl:::DRAM_ENERGY:PACKAGE0 (Power in Watts)	26.064	25.711	2	25.887	0.176
[GROUP=MAX_MARKER] rapl:::DRAM_ENERGY:PACKAGE1 (Power in Watts)	24.373	23.965	4	24.232	0.159
[GROUP=MAX_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE0 (Power in Watts)	126.872	125.182	6	125.732	0.557
[GROUP=MAX_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE1 (Power in Watts)	124.377	116.689	5	122.428	2.885
[GROUP=MAX_MARKER] rapl:::PP0_ENERGY:PACKAGE0 (Power in Watts)	103.981	102.21	6	102.769	0.584
[GROUP=MAX_MARKER] rapl:::PP0_ENERGY:PACKAGE1 (Power in Watts)	102.615	101.693	4	102.115	0.33
rapl:::DRAM_ENERGY:PACKAGE0 (Power in Watts)	26.064	15.84	36	19.053	3.39
rapl:::DRAM_ENERGY:PACKAGE1 (Power in Watts)	24.373	14.275	36	16.435	3.155
rapl:::PACKAGE_ENERGY:PACKAGE0 (Power in Watts)	126.872	93.125	36	117.729	5.403
rapl:::PACKAGE_ENERGY:PACKAGE1 (Power in Watts)	124.377	96.853	36	112.961	4.776
rapl:::PP0_ENERGY:PACKAGE0 (Power in Watts)	103.981	75.096	36	98.208	4.466
rapl:::PP0_ENERGY:PACKAGE1 (Power in Watts)	102.615	79.646	36	94.872	3.662

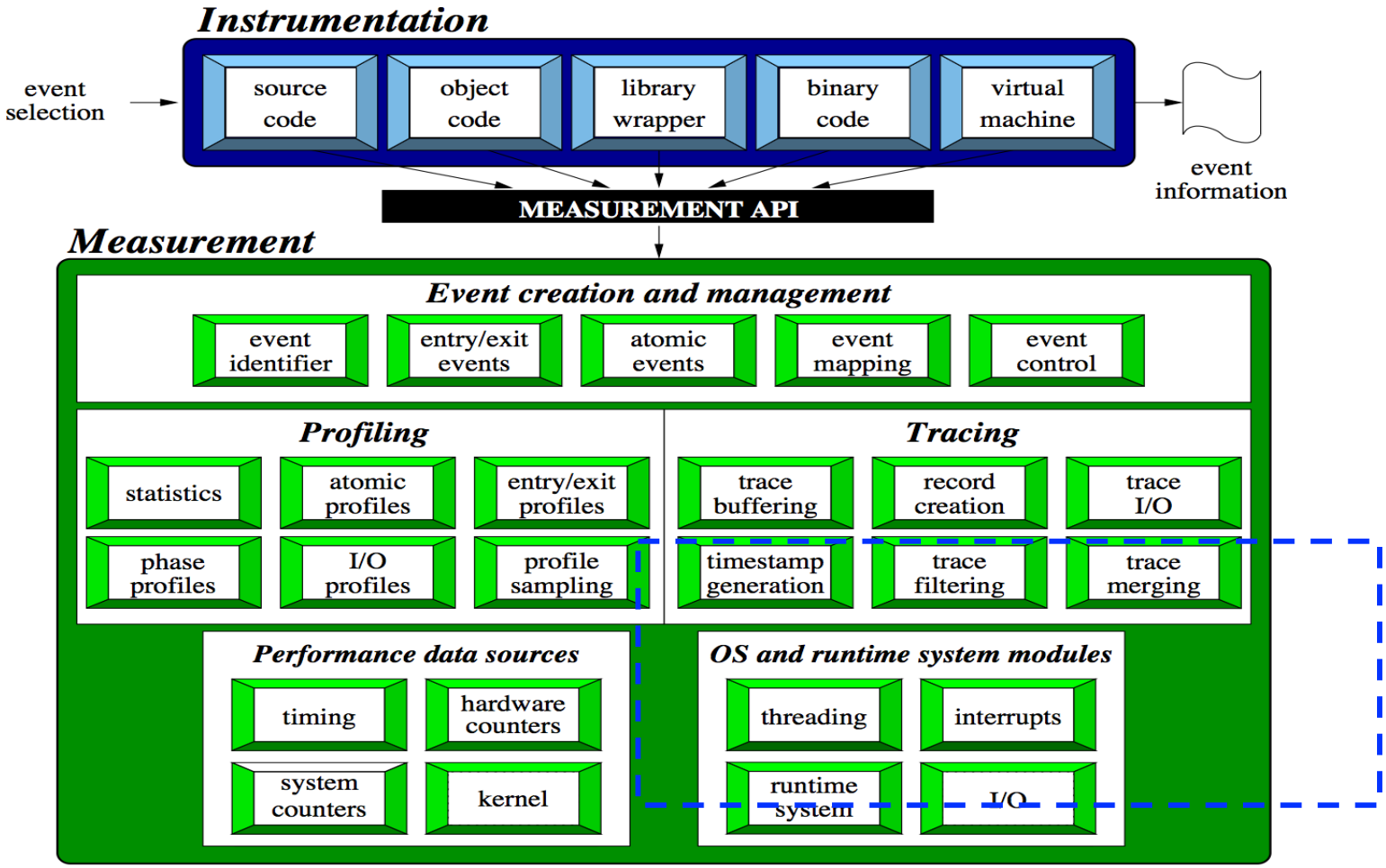
When an atomic event exceeds the max or min value by a threshold (say 20%), a marker context event is triggered to record the callstack.

Marker Events: Tracing



When an atomic event exceeds the max or min value by a threshold (say 20%), a marker context event is triggered to record the callstack.

TAU Instrumentation / Measurement



Direct Instrumentation Options in TAU

Source Code Instrumentation

- Automatic instrumentation using pre-processor based on static analysis of source code (PDT), creating an instrumented copy
- Compiler generates instrumented object code
- Manual instrumentation

Library Level Instrumentation

- Statically or dynamically linked wrapper libraries
  - MPI, I/O, memory, etc.
- Wrapping external libraries where source is not available

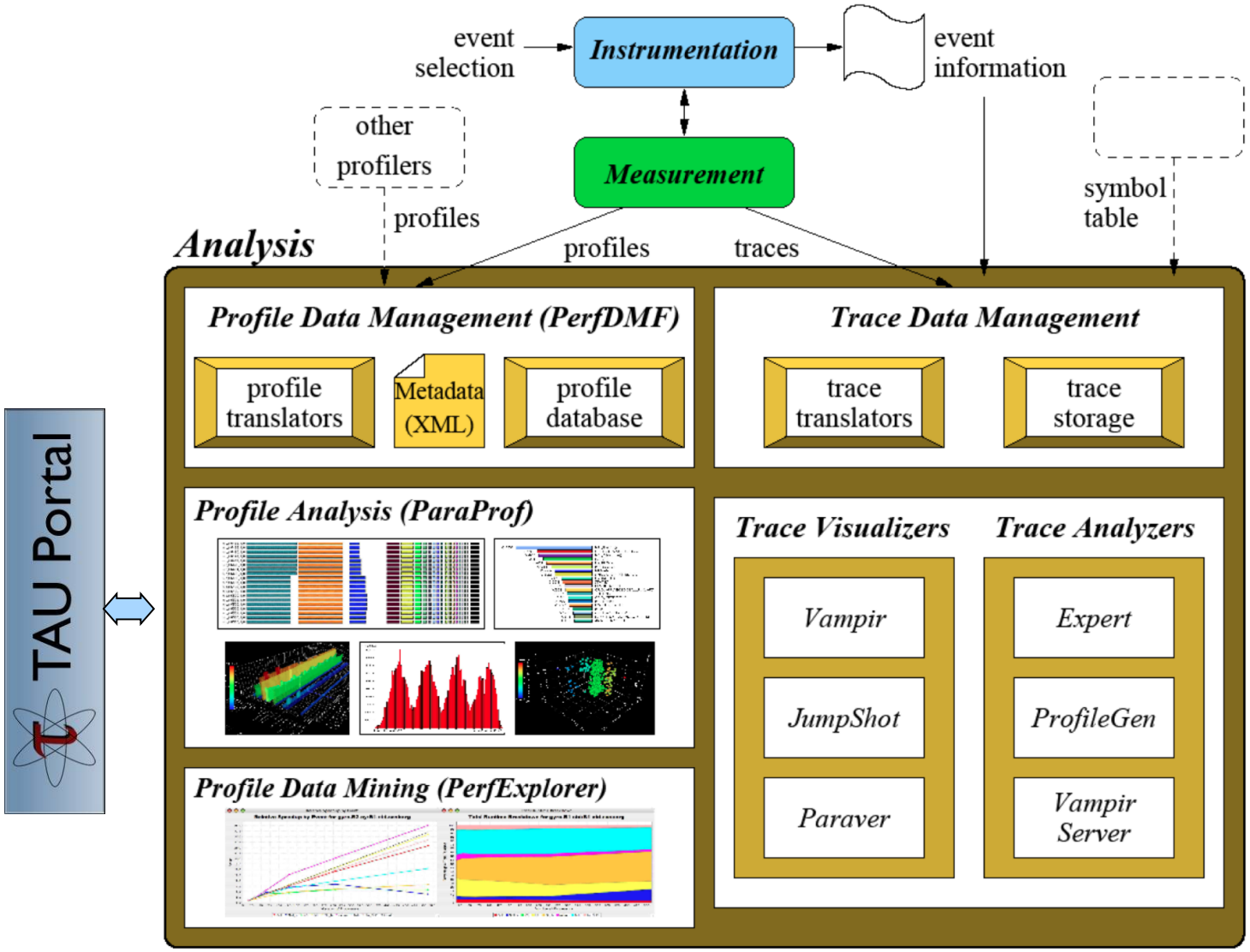
Runtime pre-loading and interception of library calls

Binary Code instrumentation

- Rewrite the binary, runtime instrumentation

Virtual Machine, Interpreter, OS level instrumentation

TAU Analysis



Direct Instrumentation Options in TAU

Source Code Instrumentation

- Manual instrumentation
- Automatic instrumentation using pre-processor based on static analysis of source code (PDT), creating an instrumented copy
- Compiler generates instrumented object code

Library Level Instrumentation

- Wrapper libraries for standard MPI libraries using PMPI interface
- Wrapping external libraries where source is not available

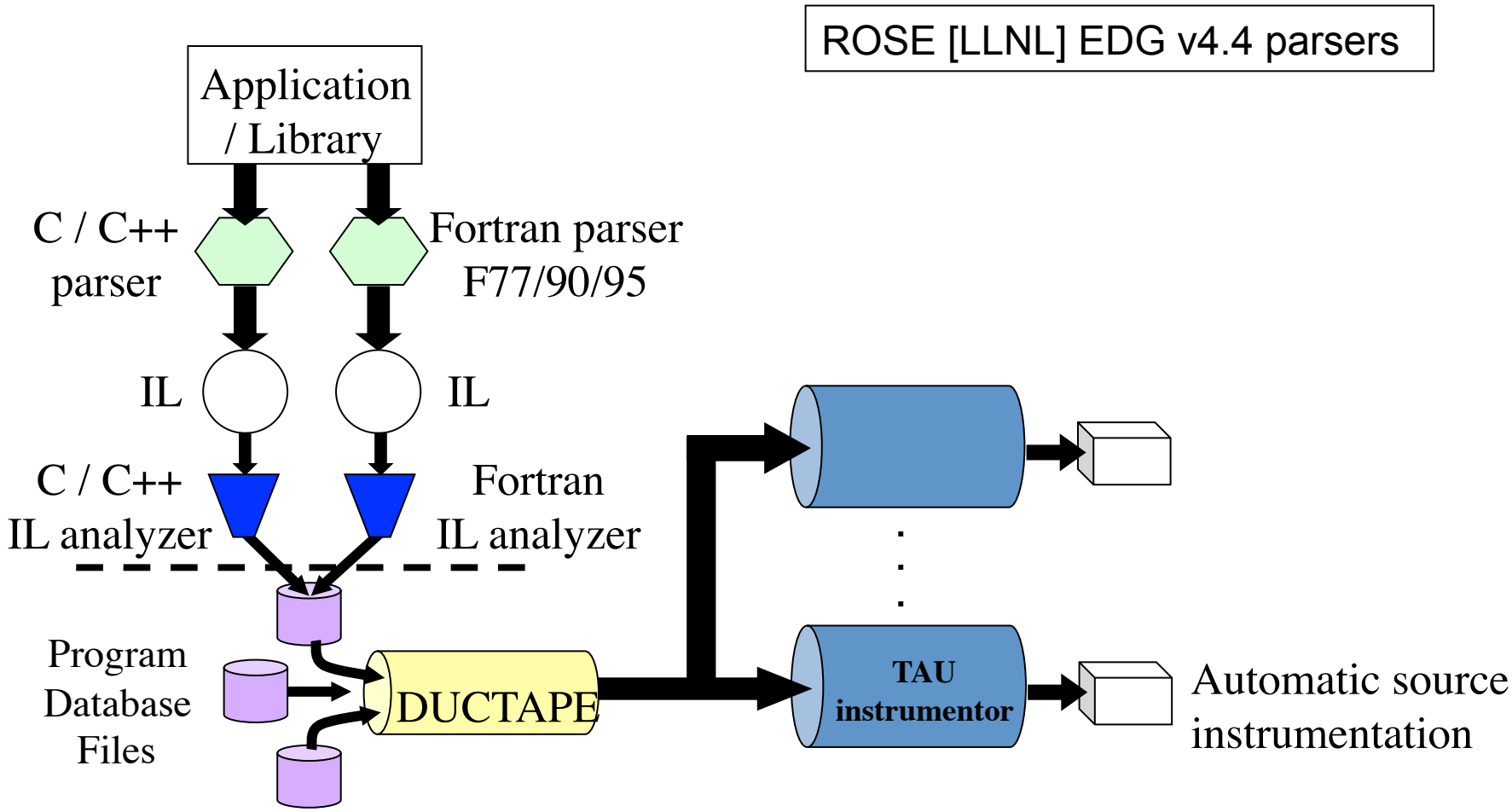
Runtime pre-loading and interception of library calls

Binary Code instrumentation

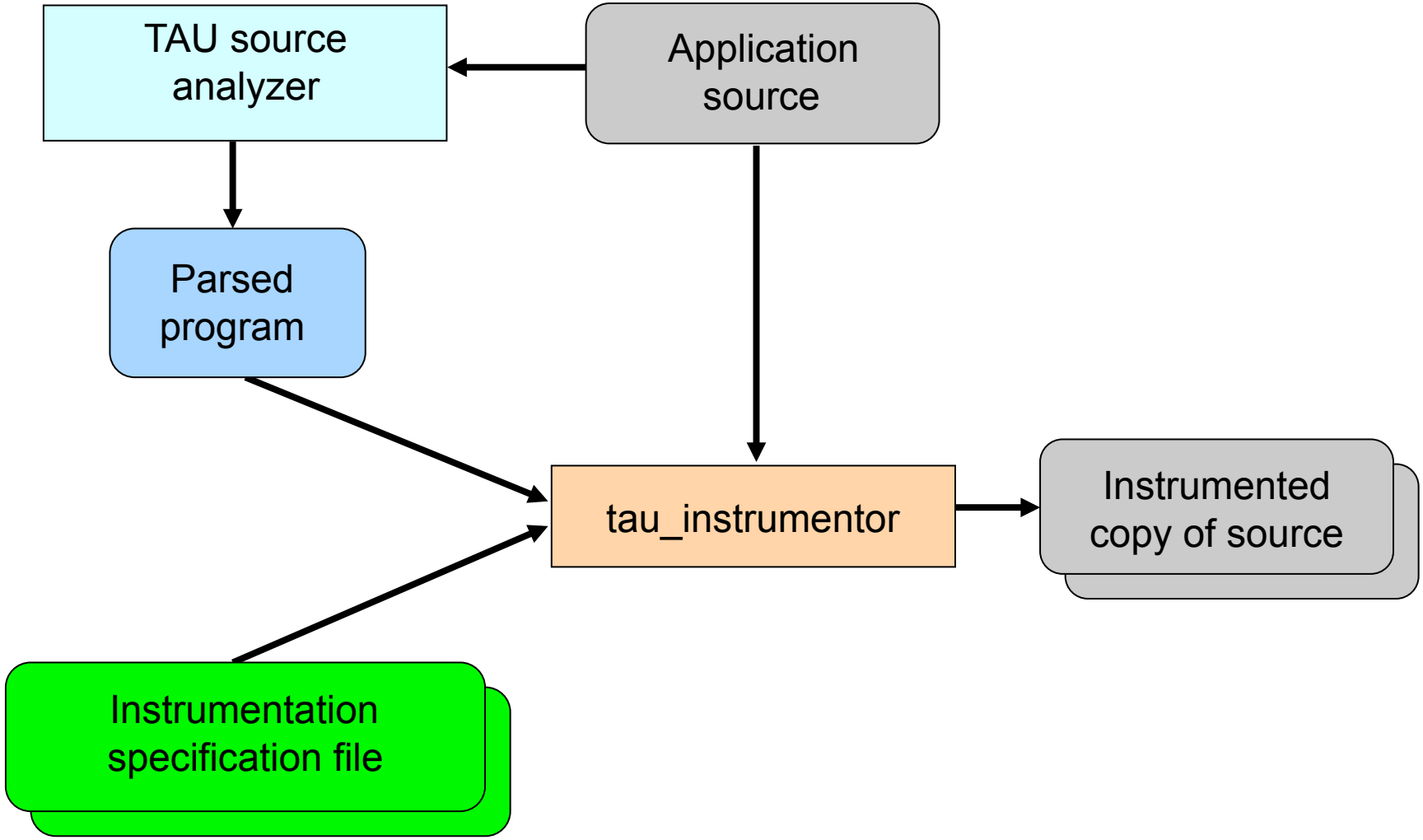
- Rewrite the binary, runtime instrumentation

Virtual Machine, Interpreter, OS level instrumentation

TAU's Static Analysis System: Program Database Toolkit (PDT)

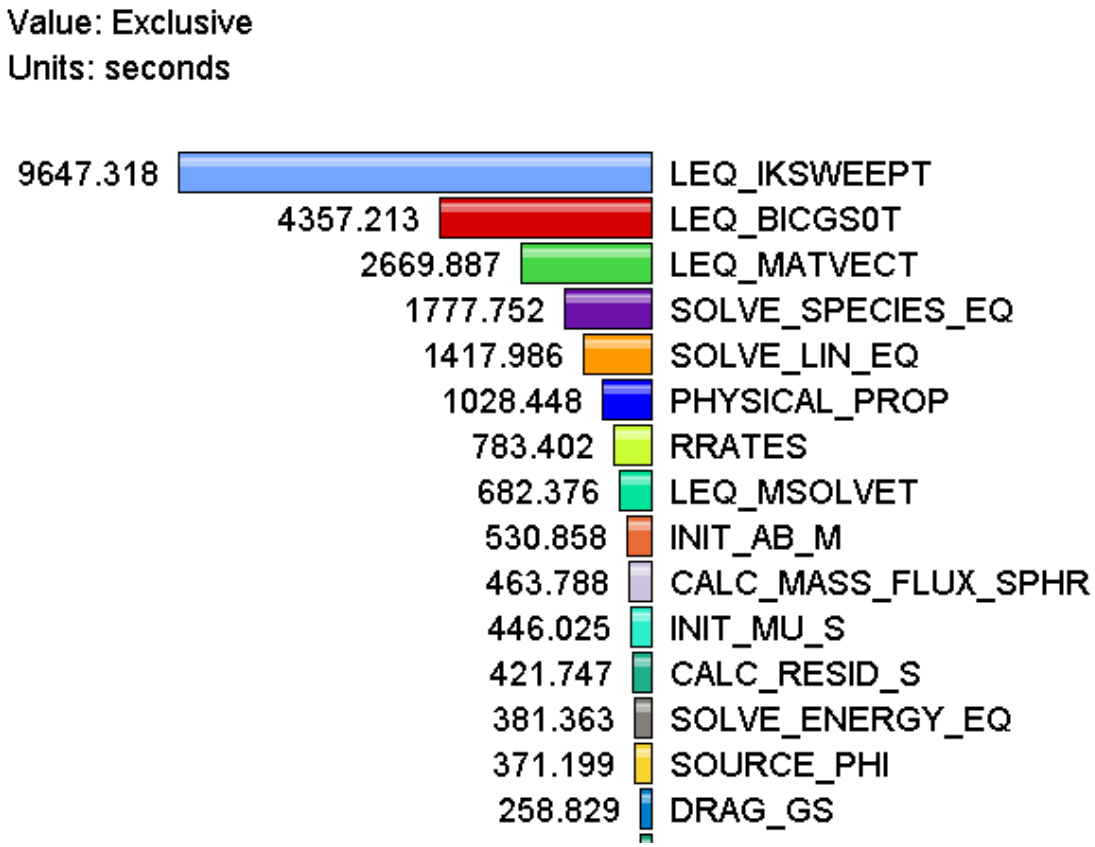


Automatic Source Instrumentation using PDT



Usage Scenarios: Routine Level Profile

How much time is spent in each application routine?



Generating a flat profile with MPI

```
% export TAU_MAKEFILE=/phihome/sameer/pkgs/tau_latest/x86_64/lib/
Makefile.tau-icpc-papi-mpi-pdt
% export PATH=$TAU/bin:$PATH
Or
% source /phihome/sameer/pkgs/tau.x86_64.bashrc(or .cshrc)
% make F90=tau_f90.sh
Or
% tau_f90.sh matmult.f90
% qsub -I -l select:ncpus=4; mpiexec ./a.out
% paraprof
To view. To view the data locally on the workstation,
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk

Click on the "node 0" label to see profile for that node. Right
click to see other options. Windows -> 3D Visualization for 3D
window.
```

Automatic Instrumentation

We now provide compiler wrapper scripts

- Simply replace `CXX` with `tau_cxx.sh`
- Automatically instruments C++ and C source code, links with TAU MPI Wrapper libraries.

Use `tau_cc.sh` and `tau_f90.sh` for C and Fortran

Before

```
CXX = mpicxx
F90 = mpif90
CXXFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o … fn.o

app: $(OBJS)
      $(CXX) $(LDFLAGS) $(OBJS) -o $@
      $(LIBS)
.cpp.o:
      $(CXX) $(CXXFLAGS) -c $<
```

After

```
CXX = tau_cxx.sh
F90 = tau_f90.sh
CXXFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o … fn.o

app: $(OBJS)
      $(CXX) $(LDFLAGS) $(OBJS) -o $@
      $(LIBS)
.cpp.o:
      $(CXX) $(CXXFLAGS) -c $<
```

Hands-on:

```
% source /phihome/sameer/pkgs/tau.x86_64.bashrc
% tar zxf /phihome/pkgs/tar/workshop.tgz
% cd matmult
% export TAU_MAKEFILE=
      $TAU/Makefile.tau-icpc-papi-mpi-pdt
% make
```

```
Edit run.job to put in correct project id in -A:
% qsub run.job
% pprof
% paraprof -pack foo.ppk
% paraprof foo.ppk   (either remotely or locally)
```

Passing Optional Parameters to TAU Compiler Scripts

See `$TAU/bin/tau_compiler.sh -help`

On Crays, compilation:

```
% ftn -c foo.f90
```

Changes to

```
% gfparse foo.f90 $(OPT1)
% tau_instrumentor foo.pdb foo.f90 -o foo.inst.f90 $(OPT2)
% ftn -c foo.inst.f90 -o foo.o $(OPT3)
```

Linking:

```
% ftn foo.o bar.o -o app
```

Changes to

```
% ftn foo.o bar.o -o app <taulibs> $(OPT4)
```

Where options OPT[1-4] default values may be overridden by the user:

```
F90 = tau_f90.sh
```

Compile-Time Options

Optional parameters for the TAU_OPTIONS environment variable:
% tau_compiler.sh

-optVerbose	Turn on verbose debugging messages
-optCompInst	Use compiler based instrumentation
-optNoCompInst	Do not revert to compiler instrumentation if source instrumentation fails.
-optTrackIO	Wrap POSIX I/O call and calculates vol/bw of I/O operations (Requires TAU to be configured with –*iowrapper*)
-optTrackGOMP	Enable tracking GNU OpenMP runtime layer (used without –opari)
-optMemDbg	Enable runtime bounds checking (see TAU_MEMDBG_* env vars)
-optKeepFiles	Does not remove intermediate .pdb and .inst.* files
-optPreProcess	Preprocess sources (OpenMP, Fortran) before instrumentation
-optTauSelectFile=”<file>"	Specify selective instrumentation file for *tau_instrumentor*
-optTauWrapFile=”<file>"	Specify path to *link_options.tau* generated by *tau_gen_wrapper*
-optHeaderInst	Enable Instrumentation of headers
-optTrackUPCR	Track UPC runtime layer routines (used with tau_upc.sh)
-optLinking=""	Options passed to the linker. Typically $(TAU_MPI_FLIBS) $(TAU_LIBS) $(TAU_CXXLIBS)
-optCompile=""	Options passed to the compiler. Typically $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
-optPdtF95Opts=""	Add options for Fortran parser in PDT (f95parse/gfparse) …

Compile-Time Options (contd.)

Optional parameters for the TAU_OPTIONS environment variable:
% tau_compiler.sh

-optMICOffload	Links code for Intel MIC offloading, requires both host and MIC TAU libraries
-optShared	Use TAU's shared library (libTAU.so) instead of static library (default)
-optPdtCxxOpts=""	Options for C++ parser in PDT (cxxparse).
-optPdtF90Parser=""	Specify a different Fortran parser
-optPdtCleanscapeParser	Specify the Cleanscape Fortran parser instead of GNU gfparser
-optTau=""	Specify options to the tau_instrumentor
-optTrackDMAPP	Enable instrumentation of low-level DMAPP API calls on Cray
-optTrackPthread	Enable instrumentation of pthread calls

See tau_compiler.sh for a full list of TAU_OPTIONS.

…

Compiling Fortran Codes with TAU

If your Fortran code uses free format in .f files (fixed is default for .f), you may use:

% export TAU_OPTIONS= '-optPdtF95Opts="-R free" -optVerbose '

To use the compiler based instrumentation instead of PDT (source-based):

% export TAU_OPTIONS= '-optCompInst -optVerbose'

If your Fortran code uses C preprocessor directives (#include, #ifdef, #endif):

% export TAU_OPTIONS= '-optPreProcess -optVerbose -optDetectMemoryLeaks'

To use an instrumentation specification file:

% export TAU_OPTIONS= '-optTauSelectFile=select.tau -optVerbose -optPreProcess'

% cat select.tau

```
BEGIN_INSTRUMENT_SECTION
loops  routine="#"
# this statement instruments all outer loops in all routines. # is wildcard as well as comment in first column.
END_INSTRUMENT_SECTION
```

Runtime Environment Variables

Environment Variable	Default	Description
TAU_TRACE	0	Setting to 1 turns on tracing
TAU_CALLPATH	0	Setting to 1 turns on callpath profiling
TAU_TRACK_MEMORY_LEAKS	0	Setting to 1 turns on leak detection (for use with –optMemDbg or tau_exec)
TAU_MEMDBG_PROTECT_ABOVE	0	Setting to 1 turns on bounds checking for dynamically allocated arrays. (Use with –optMemDbg or tau_exec –memory_debug).
TAU_CALLPATH_DEPTH	2	Specifies depth of callpath. Setting to 0 generates no callpath or routine information, setting to 1 generates flat profile and context events have just parent information (e.g., Heap Entry: foo)
TAU_SAMPLING	1	Setting to 1 enables event-based sampling.
TAU_TRACK_SIGNALS	0	Setting to 1 generate debugging callstack info when a program crashes
TAU_COMM_MATRIX	0	Setting to 1 generates communication matrix display using context events
TAU_THROTTLE	1	Setting to 0 turns off throttling. Enabled by default to remove instrumentation in lightweight routines that are called frequently
TAU_THROTTLE_NUMCALLS	100000	Specifies the number of calls before testing for throttling
TAU_THROTTLE_PERCALL	10	Specifies value in microseconds. Throttle a routine if it is called over 100000 times and takes less than 10 usec of inclusive time per call
TAU_COMPENSATE	0	Setting to 1 enables runtime compensation of instrumentation overhead
TAU_PROFILE_FORMAT	Profile	Setting to “merged” generates a single file. “snapshot” generates xml format
TAU_METRICS	TIME	Setting to a comma separated list generates other metrics. (e.g., TIME,P_VIRTUAL_TIME,PAPI_FP_INS,PAPI_NATIVE_<event>:<subevent>)

Runtime Environment Variables (contd.)

Environment Variable	Default	Description
TAU_TRACK_MEMORY_LEAKS	0	Tracks allocates that were not de-allocated (needs –optMemDbg or tau_exec –memory)
TAU_EBS_SOURCE	TIME	Allows using PAPI hardware counters for periodic interrupts for EBS (e.g., TAU_EBS_SOURCE=PAPI_TOT_INS when TAU_SAMPLING=1)
TAU_EBS_PERIOD	100000	Specifies the overflow count for interrupts
TAU_MEMDBG_ALLOC_MIN/MAX	0	Byte size minimum and maximum subject to bounds checking (used with TAU_MEMDBG_PROTECT_*)
TAU_MEMDBG_OVERHEAD	0	Specifies the number of bytes for TAU's memory overhead for memory debugging.
TAU_MEMDBG_PROTECT_BELOW/ ABOVE	0	Setting to 1 enables tracking runtime bounds checking below or above the array bounds (requires –optMemDbg while building or tau_exec –memory)
TAU_MEMDBG_ZERO_MALLOC	0	Setting to 1 enables tracking zero byte allocations as invalid memory allocations.
TAU_MEMDBG_PROTECT_FREE	0	Setting to 1 detects invalid accesses to deallocated memory that should not be referenced until it is reallocated (requires –optMemDbg or tau_exec –memory)
TAU_MEMDBG_ATTEMPT_CONTINUE	0	Setting to 1 allows TAU to record and continue execution when a memory error occurs at runtime.
TAU_MEMDBG_FILL_GAP	Undefined	Initial value for gap bytes
TAU_MEMDBG_ALINGMENT	Sizeof(int)	Byte alignment for memory allocations
TAU_EVENT_THRESHOLD	0.5	Define a threshold value (e.g., .25 is 25%) to trigger marker events for min/ max

Loop Level Instrumentation

Goal: What loops account for the most time? How much?

Flat profile with wallclock time with loop instrumentation:

Metric: GET_TIME_OF_DAY
Value: Exclusive
Units: microseconds

1729975.333	Loop: MULTIPLY_MATRICES [{matmult.f90} {31,9}-{36,14}]
443194	MPI_Recv()
81095	MAIN
49569	MPI_Bcast()
45669	Loop: MAIN [{matmult.f90} {86,9}-{106,14}]
12412	MPI_Send()
8959	Loop: INITIALIZE [{matmult.f90} {17,9}-{21,14}]
8953	Loop: INITIALIZE [{matmult.f90} {10,9}-{14,14}]
5609.2	MPI_Finalize()
2932.667	MULTIPLY_MATRICES
2577.667	Loop: MAIN [{matmult.f90} {117,9}-{128,14}]
2091.8	MPI_Barrier()
1875.667	Loop: MAIN [{matmult.f90} {112,9}-{115,14}]
1833	Loop: MAIN [{matmult.f90} {71,9}-{74,14}]
107	Loop: MAIN [{matmult.f90} {77,9}-{84,14}]
30	INITIALIZE
14.25	MPI_Comm_rank()
1	MPI_Comm_size()

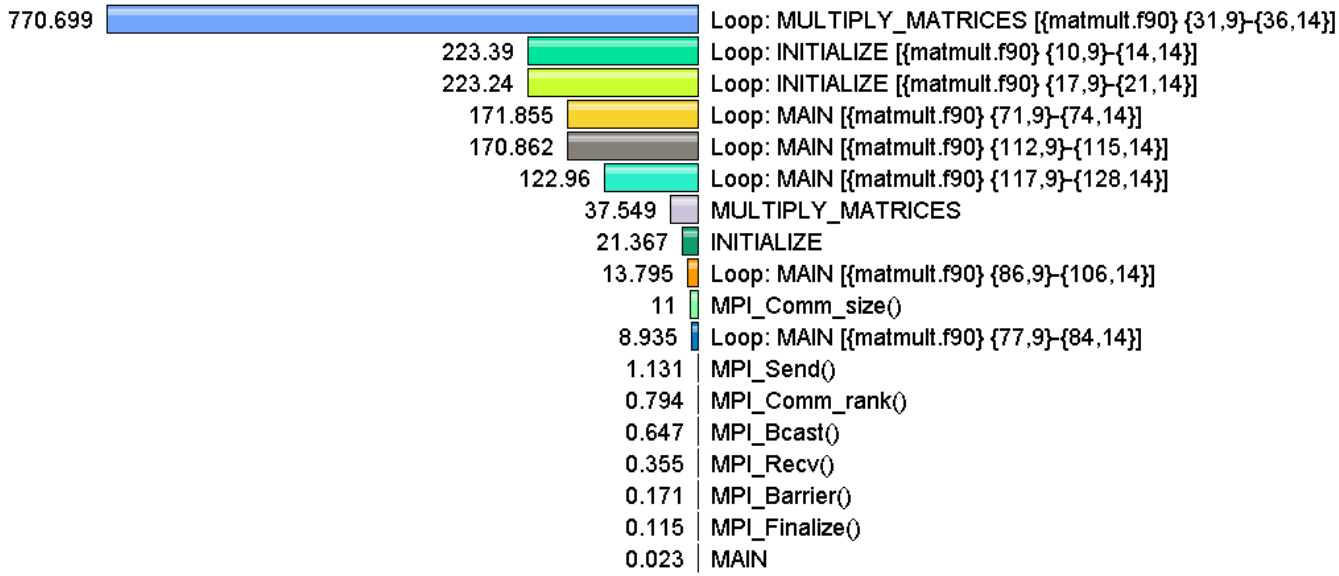
Generating a loop level profile

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% export TAU_OPTIONS='-optTauSelectFile=select.tau -optVerbose'
% cat select.tau
  BEGIN_INSTRUMENT_SECTION
  loops routine="#"
  END_INSTRUMENT_SECTION


% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% mxterm 1 4 40 ; mpirun -np 4  ./matmult
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.

% paraprof app.ppk
```

Computing FLOPS for loops

Goal: What execution rate do my application loops get in mflops?

Metric: PAPI_FP_INS / GET_TIME_OF_DAY
Value: Exclusive
Units: Derived metric shown in microseconds format

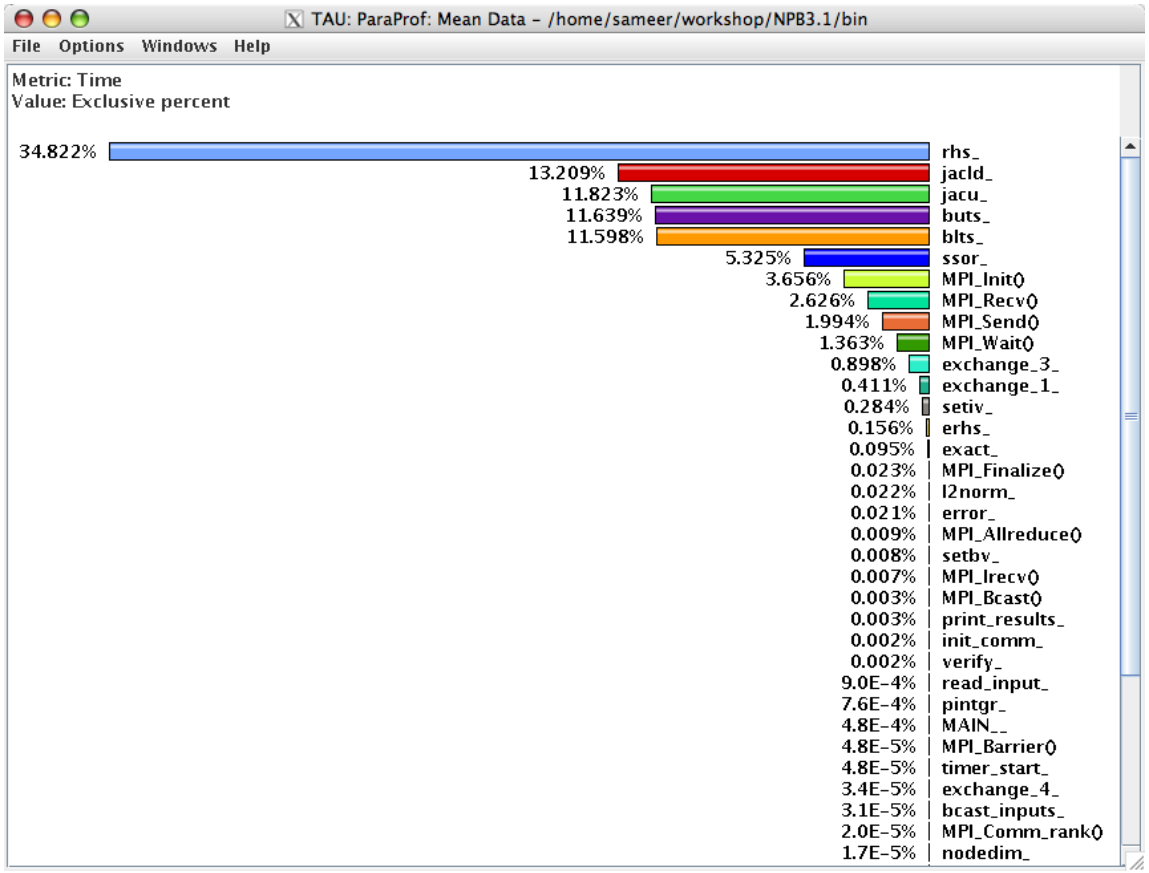


Profiling with multiple counters

```
% source /phihome/sameer/pkgs/tau.x86_64.bashrc
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% export TAU_OPTIONS='-optTauSelectFile=select.tau -optVerbose'
% cat select.tau
  BEGIN_INSTRUMENT_SECTION
  loops routine="#"
  END_INSTRUMENT_SECTION

% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% qsub -I -l ncpus=4; module load mpi
% export TAU_METRICS=TIME:PAPI_FP_INS:PAPI_L1_DCM
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
  Choose Options -> Show Derived Panel -> Click PAPI_FP_INS,
 Click "/",  Click TIME, Apply, Choose new metric by double
clicking.
```

Compiler-based Instrumentation

Use the compiler to automatically emit instrumentation calls in the object code instead of parsing the source code using PDT.



Use Compiler-Based Instrumentation

```
% source /phihome/sameer/pkgs/tau.x86_64.bashrc
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% export TAU_OPTIONS='-optCompInst -optQuiet'

% make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh

NOTE: You may also use the short-hand scripts taucc, tauf90,
taucxx instead of specifying TAU_OPTIONS and using the
traditional tau_<cc,cxx,f90>.sh scripts. These scripts use
compiler-based instrumentation by default.

% make CC=taucc CXX=taucxx F90=tauf90
% qsub -I -l ncpus=4; module load mpi
% mpiexec./a.out
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
```

Generate a Callpath Profile

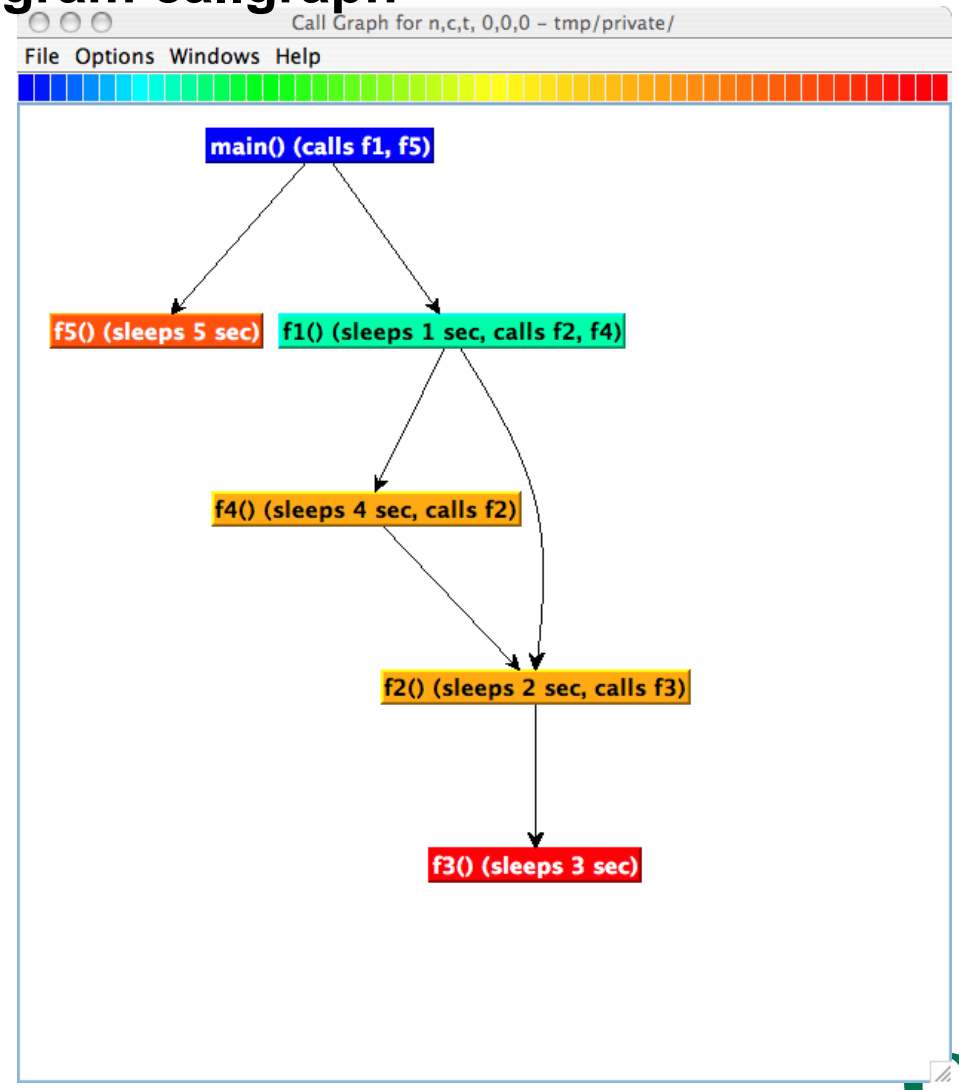
TAU: ParaProf: node 0 - profile.cubex

Metric: Time
Value: Exclusive
Units: milliseconds

319.665	int main(int, char **) C => MPI_Init
319.665	MPI_Init
278.297	int main(int, char **) C => double do_work(void) C => void compute(double **, double **, double **, int, int, int) C
278.297	void compute(double **, double **, double **, int, int, int) C
264.864	int main(int, char **) C => double do_work(void) C => void compute_interchange(double **, double **, double **, int,
264.864	void compute_interchange(double **, double **, double **, int, int, int) C
244.177	int main(int, char **) C => MPI_Finalize
244.177	MPI_Finalize
3.327	int main(int, char **) C => double do_work(void) C => void initialize(double **, int, int) C
3.327	void initialize(double **, int, int) C
2.601	int main(int, char **) C => double do_work(void) C => double **allocateMatrix(int, int) C
2.601	double **allocateMatrix(int, int) C
1.173	int main(int, char **) C => double do_work(void) C
1.173	double do_work(void) C
0.185	int main(int, char **) C

Callpath Profile

Generates program callgraph

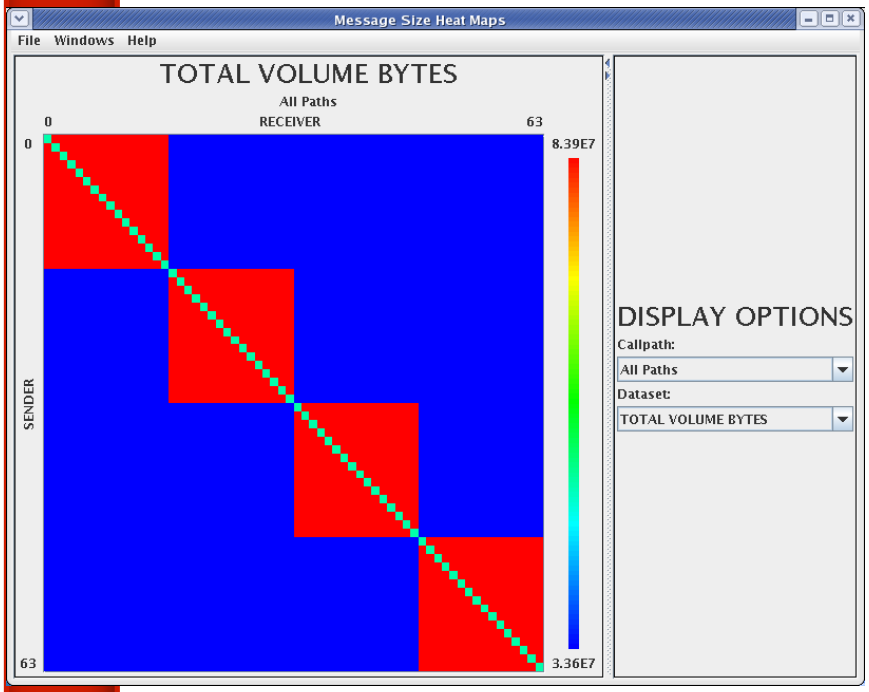


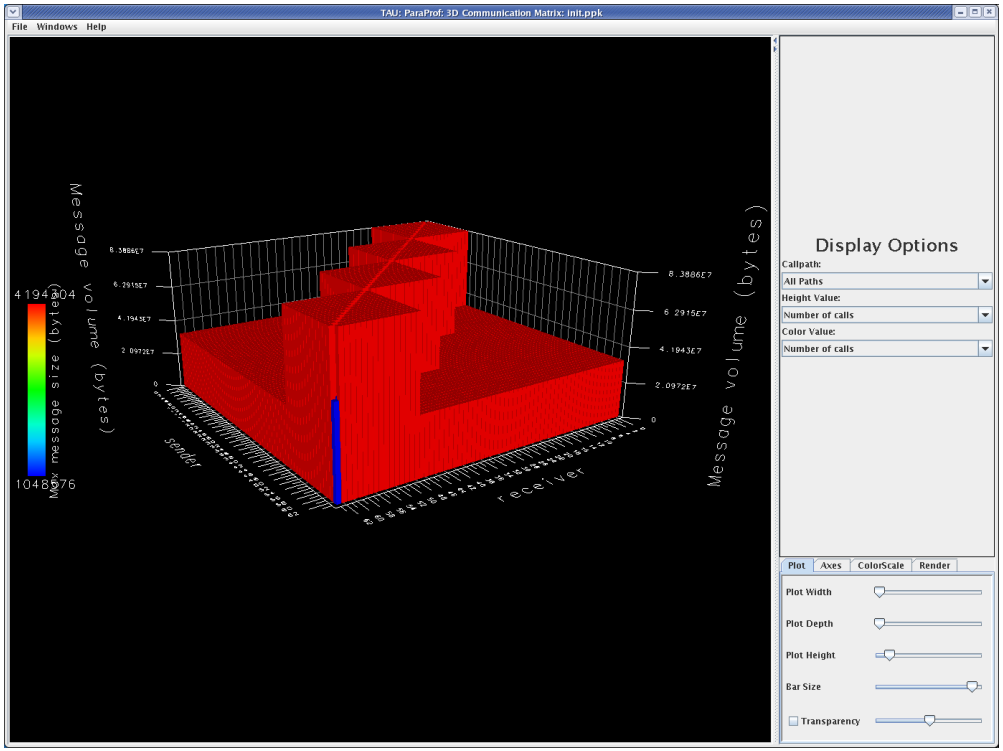
Generate a Callpath Profile

```
% source /phihome/sameer/pkgs/tau.x86_64.bashrc
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% export PATH=$TAU/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)

% qsub -I -l ncpus=4
% export TAU_CALLPATH=1
% export TAU_CALLPATH_DEPTH=100
(truncates all calling paths to a specified depth)
% mpiexec  ./a.out
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
(Windows -> Thread -> Call Graph)
```

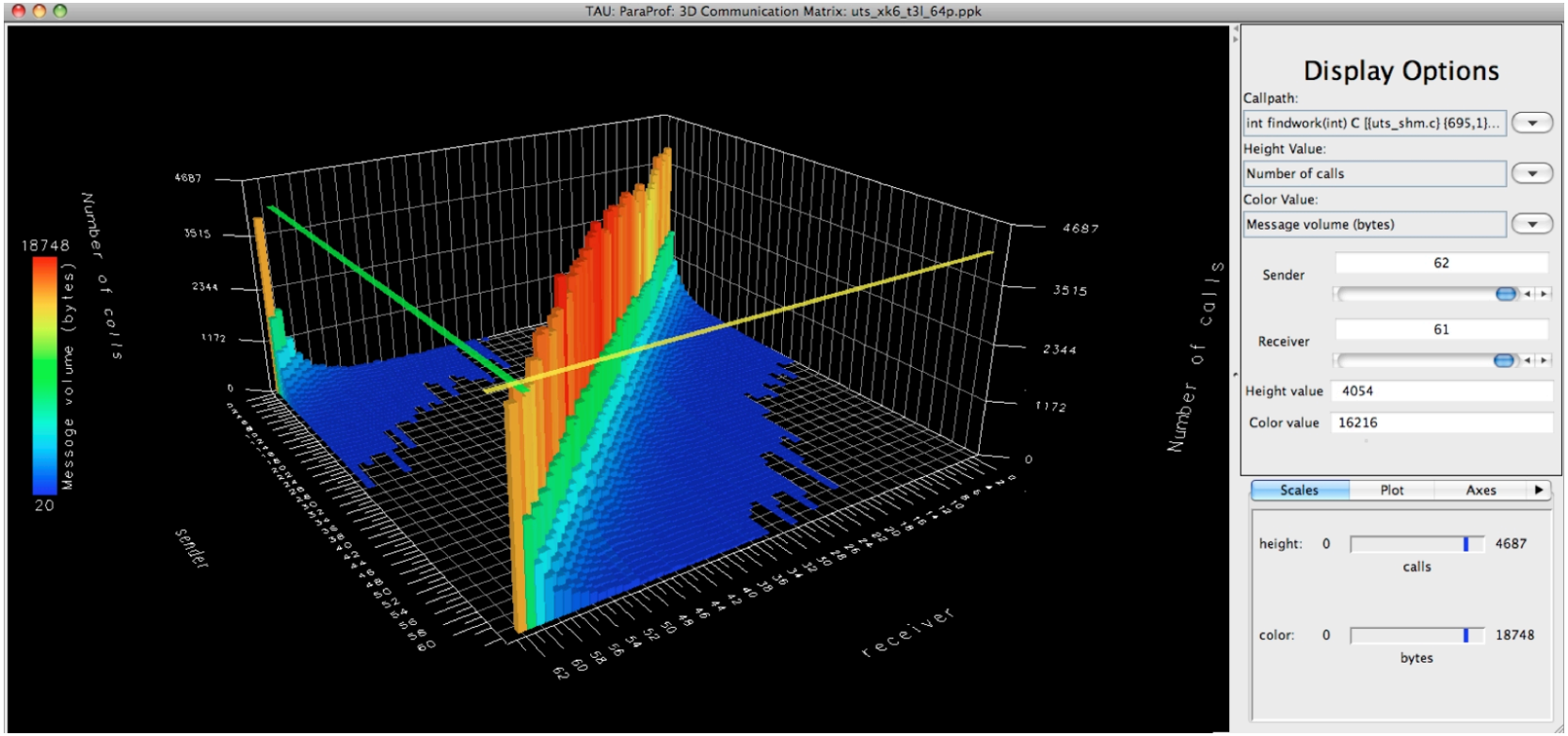
Communication Matrix Display

Goal: What is the volume of inter-process communication? Along which calling path?

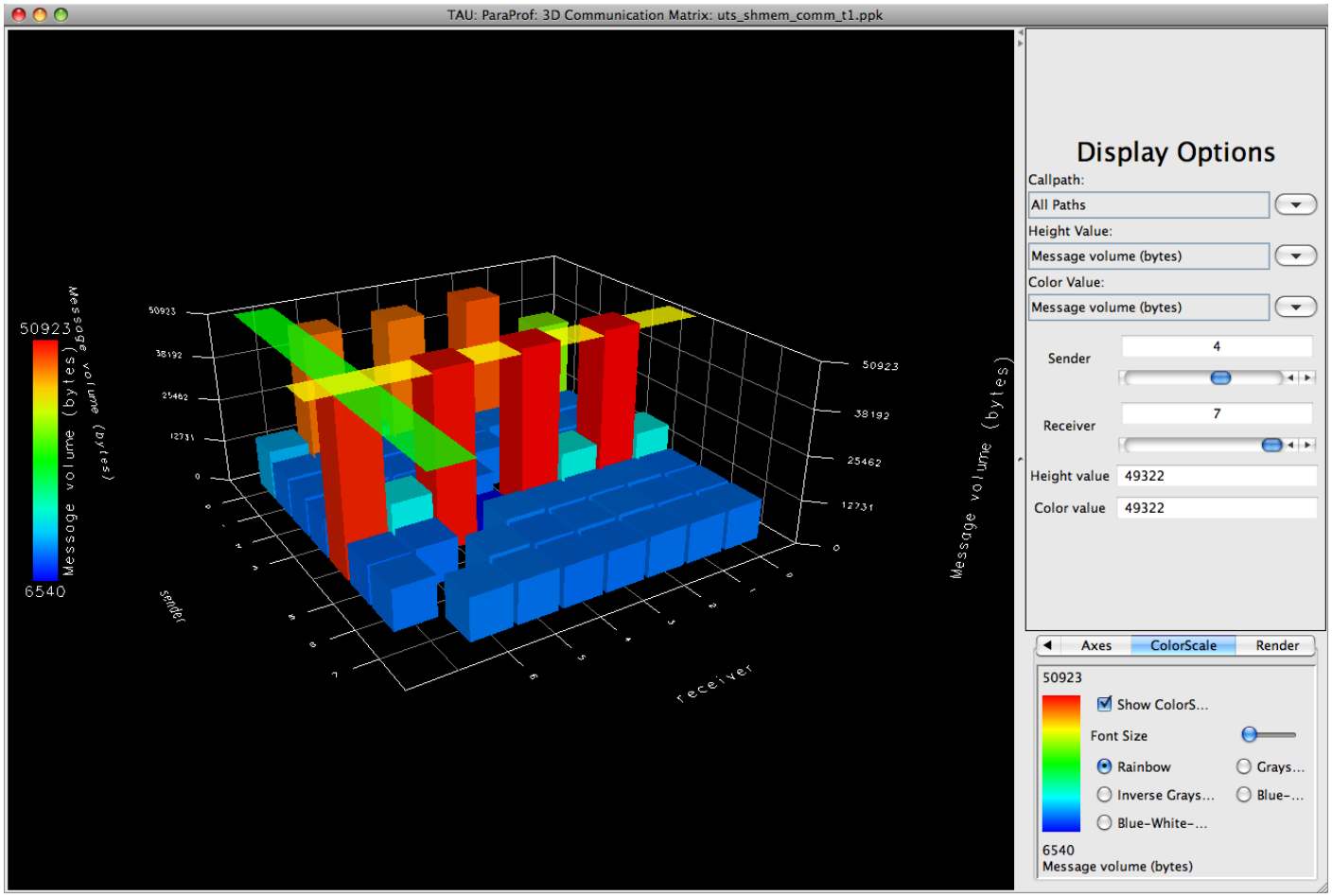




ParaProf: 3D Communication Matrix



SHMEM Communication Matrix



ParaTools
UNIVERSITY OF OREGON

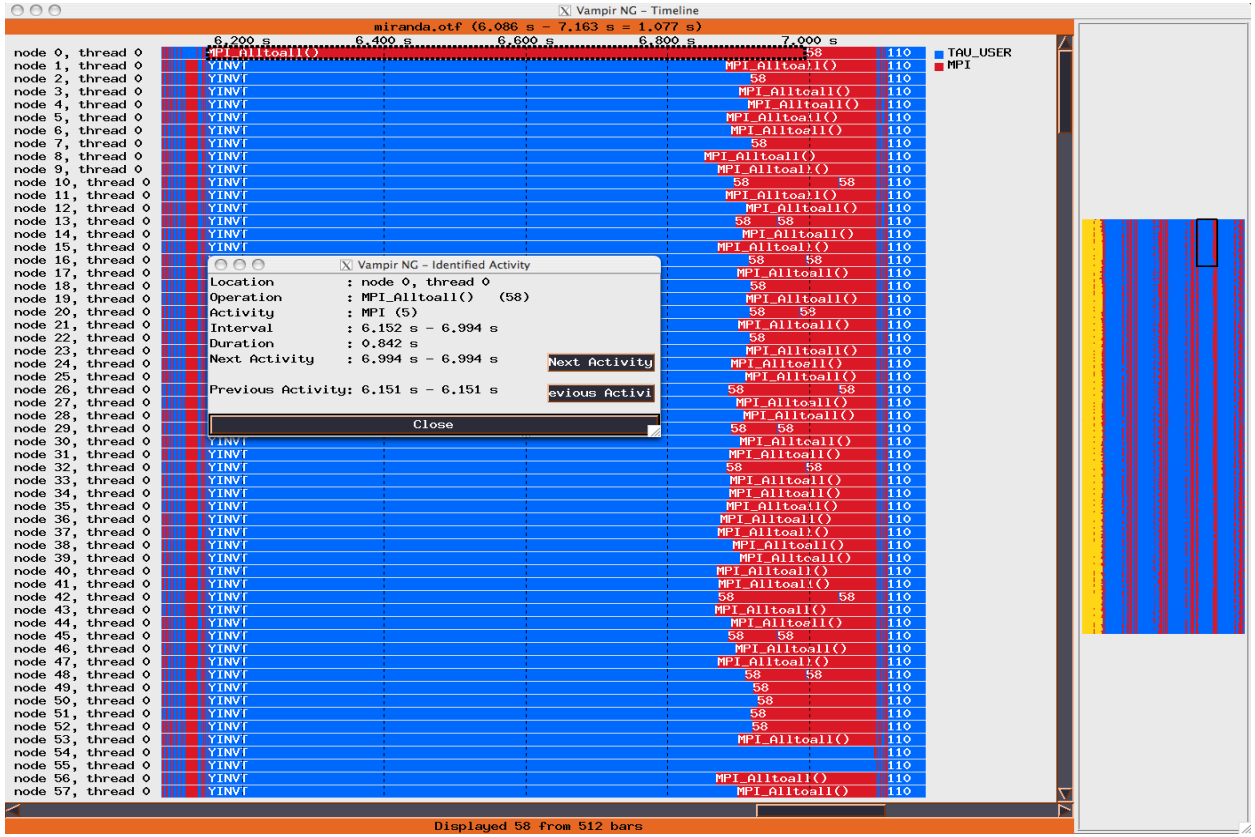
Generating Communication Matrix

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% export PATH=$TAU/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)

% qsub -I -l ncpus=4;
% export TAU_COMM_MATRIX=1
% mpiexec  ./a.out

% paraprof
(Windows -> Communication Matrix)
(Windows -> 3D Communication Matrix)
```

Generating a Trace File

Goal: Identify the temporal aspect of performance. What happens in my code at a given time? When?

Event trace visualized in Vampir/Jumpshot/Paraver



ParaTools
UNIVERSITY OF OREGON

Generating Event Traces

```
% source /phihome/sameer/pkgs/tau.x86_64.bashrc
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% export PATH=$TAU/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)

% qsub -I -l ncpus=4
% export TAU_TRACE=1
% mpiexec  ./a.out

Merge and convert the travefiles:
% tau_treemerge.pl
For Vampir (OTF):
% tau2otf tau.trc tau.edf app.otf; vampir app.otf

For Jumpshot (SLOG2):
% tau2slog2 tau.trc tau.edf -o app.slog2; jumpshot app.slog2

For ParaVer:
% tau_convert -paraver tau.trc tau.edf app.prv; paraver app.prv
```

Three Instrumentation Techniques for Wrapping External Libraries

Pre-processor based substitution by re-defining a call (e.g., read)

- Tool defined header file with same name <*unistd.h*> takes precedence
- Header redefines a routine as a different routine using macros
- Substitution: *read()* substituted by preprocessor as *tau_read()* at callsite

Preloading a library at runtime

- Library preloaded (*LD_PRELOAD* env var in Linux) in the address space of executing application intercepts calls from a given library
- Tool's wrapper library defines *read(),* gets address of global *read()* symbol (dlsym), internally calls timing calls around call to global read

Linker based substitution

- Wrapper library defines *__wrap_read* which calls *__real_read* and linker is passed *-Wl,-wrap,read* to substitute all references to read from application's object code with the *__wrap_read* defined by the tool

Issues: Preprocessor based substitution

Pre-processor based substitution by re-defining a call

- Compiler replaces read() with tau_read() in the body of the source code

Advantages:

- Simple to instrument
  - Preprocessor based replacement
  - A header file redefines the calls
  - No special linker or runtime flags required

Disadvantages

- Only works for C & C++ for replacing calls in the body of the code.
- Incomplete instrumentation: fails to capture calls in uninstrumented libraries (e.g., libhdf5.a)

Issues: Linker based substitution

Linker based substitution

- Wrapper library defines __wrap_read which calls __real_read and linker is passed -Wl,-wrap, read

Advantages

- Tool can intercept all references to a given call
- Works with static as well as dynamic executables
- No need to recompile the application source code, just re-link the application objects and libraries with the tool wrapper library

Disadvantages

- Wrapping an entire library can lengthen the linker command line with multiple –Wl,-wrap,<func> arguments. It is better to store these arguments in a file and pass the file to the linker
- Approach does not work with un-instrumented binaries

tau_gen_wrapper

Automates creation of wrapper libraries using TAU

Input:

- header file (foo.h)
- library to be wrapped (/path/to/libfoo.a)
- technique for wrapping
  - Preprocessor based redefinition (-d)
  - Runtime preloading (-r)
  - Linker based substitution (-w: default)
- Optional selective instrumentation file (-f select)
  - Exclude list of routines, or
  - Include list of routines

Output:

- wrapper library
- optional *link_options.tau* file (-w), pass –optTauWrapFile=<file> in TAU_OPTIONS environment variable

Design of wrapper generator (tau_gen_wrapper)

***tau_gen_wrapper* shell script:**

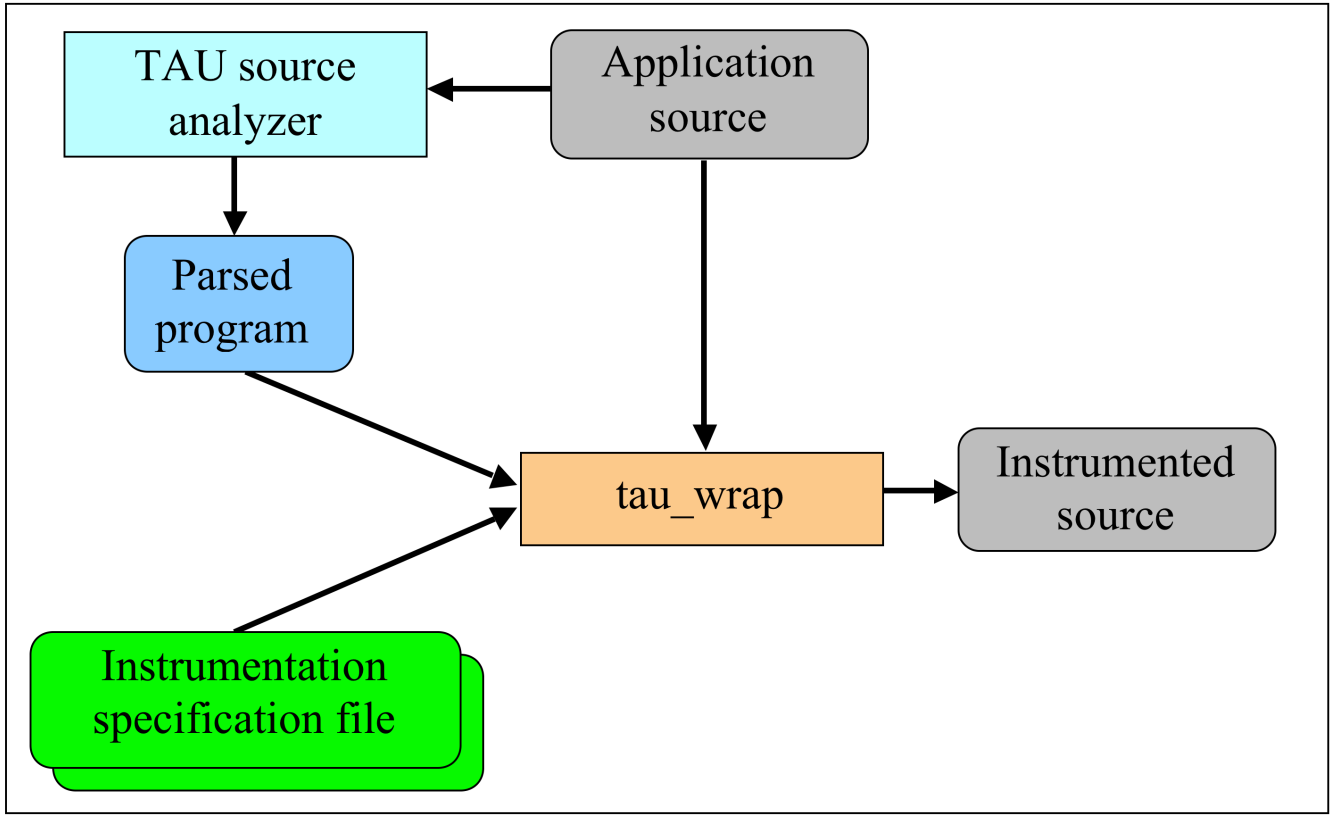
- parses source of header file using static analysis tool Program Database Toolkit (PDT)
- Invokes *tau_wrap*, a tool that generates
  - instrumented wrapper code,
  - an optional *link_options.tau* file (for linker-based substitution, -w)
  - Makefile for compiling the wrapper interposition library
- Builds the wrapper library using make

Use TAU_OPTIONS environment variable to pass location of link_options.tau file using

```
% export TAU_OPTIONS='-optTauWrapFile=<path/to/
link_options.tau> -optVerbose'
```

Use *tau_exec –loadlib=<wrapperlib.so>* to pass location of wrapper library for preloading based substitution

tau_wrap


TAU source analyzer
Application source
Parsed program
tau_wrap
Instrumented source
Instrumentation specification file

HDF5 Library Wrapping

```
[sameer@zorak]$ tau_gen_wrapper hdf5.h /usr/lib/libhdf5.a -f select.tau

Usage : tau_gen_wrapper <header> <library> [-r|-d|-w (default)] [-g
groupname] [-i headerfile] [-c|-c++|-fortran] [-f <instr_spec_file> ]
• instruments using runtime preloading (-r), or -Wl,-wrap linker (-w),
redirection of header file to redefine the wrapped routine (-d)
• instrumentation specification file (select.tau)
• group (hdf5)
• tau_exec loads libhdf5_wrap.so shared library using –
loadlib=<libwrap_pkg.so>
• creates the wrapper/ directory

NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.057            1           1          13       1236 .TAU Application
 70.8        0.875        0.875           1           0        875 hid_t H5Fcreate()
  9.7         0.12         0.12           1           0        120 herr_t H5Fclose()
  6.0        0.074        0.074           1           0         74 hid_t H5Dcreate()
  3.1        0.038        0.038           1           0         38 herr_t H5Dwrite()
  2.6        0.032        0.032           1           0         32 herr_t H5Dclose()
  2.1        0.026        0.026           1           0         26 herr_t
H5check_version()
```

Using POSIX I/O wrapper library

Setting environment variable TAU_OPTIONS=-optTrackIO links in TAU's wrapper interposition library using linker-based substitution

Instrumented application generates bandwidth, volume data

Workflow:

- % export TAU_OPTIONS= '-optTrackIO –optVerbose'
- % export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
- % make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh
- % mpirun –np 8 ./a.out
- % paraprof

Get additional data regarding individual arguments by setting environment variable TAU_TRACK_IO_PARAMS=1 prior to running

Preloading a wrapper library

Preloading a library at runtime

- Tool defines read(), gets address of global read() symbol (dlsym), internally calls timing calls around call to global read
- *tau_exec* tool uses this mechanism to intercept library calls

Advantages

- No need to re-compile or re-link the application source code
- Drop-in replacement library implemented using LD_PRELOAD environment variable under Linux, Cray CNL, IBM BG/P CNK, Solaris…

Disadvantages

- Only works with dynamic executables. Default compilation mode under Cray XE6 and IBM BG/P is to use static executables
- Not all operating systems support preloading of dynamic shared objects (DSOs)

Runtime Preloading: tau_exec

- Runtime instrumentation by pre-loading the measurement library
- Works on dynamic executables (default under Linux)
- Can substitute I/O, MPI, SHMEM, CUDA, OpenCL, and memory allocation/deallocation routines with instrumented calls
- Track interval events (e.g., time spent in write()) as well as atomic events (e.g., how much memory was allocated) in wrappers
- Accurately measure I/O and memory usage
- Preload any wrapper interposition library in the context of the executing application

Preloading a TAU Library

```
% ./configure -pdt=<dir> -mpi -papi=<dir>; make install
```

Creates in <taudir>/<arch>/lib:

```
Makefile.tau-papi-mpi-pdt
shared-papi-mpi-pdt/libTAU.so
```

```
% ./configure -pdt=<dir> -mpi; make install
```

creates

```
Makefile.tau-mpi-pdt
shared-mpi-pdt/libTAU.so
```

To explicitly choose preloading of shared-<options>/libTAU.so change:

```
% mpirun -np 8 ./a.out
```

to

```
% mpirun -np 8 tau_exec -T <comma_separated_options> ./a.out
```

```
% mpirun -np 8 tau_exec -T papi,mpi,pdt ./a.out
```

Preloads `<taudir>/<arch>/shared-papi-mpi-pdt/libTAU.so`

```
% mpirun -np 8 tau_exec -T papi ./a.out
```

Preloads `<taudir>/<arch>/shared-papi-mpi-pdt/libTAU.so` by matching.

```
% mpirun -np 8 tau_exec -T papi,mpi,pdt -s ./a.out
```

Does not execute the program. Just displays the library that it will preload if executed without the **–s** option.

NOTE: -mpi configuration is selected by default. Use –T serial for Sequential programs.

TAU Execution Command (tau_exec)

Uninstrumented execution

- % mpirun –np 8 ./a.out

Track MPI performance

- % mpirun –np 8 tau_exec ./a.out

Track POSIX I/O and MPI performance (MPI enabled by default)

- % mpirun –np 8 tau_exec –io ./a.out

Track memory operations

- % setenv TAU_TRACK_MEMORY_LEAKS 1
- % mpirun –np 8 tau_exec –memory ./a.out

Use event based sampling (compile with –g)

- % mpirun –np 8 tau_exec –ebs ./a.out
- Also –ebs_source=<PAPI_COUNTER> -ebs_period=<overflow_count>

Load wrapper interposition library

- % mpirun –np 8 tau_exec –loadlib=<path/libwrapper.so> ./a.out

Track GPGPU operations

- % mpirun –np 8 tau_exec –cuda ./a.out
- % mpirun –np 8 tau_exec –opencl ./a.out

Tags in tau_exec and other tools

```
% cd $TAU; ls Makefile.*
Makefile.tau-icpc-papi-mpi-pdt

% qsub -I -X -l walltime=0:30:00 -l ncpus=4
% source /phihome/sameer/pkgs/tau.x86_64.bashrc(or .cshrc)
% tau_exec -T icpc,mpi,pdt./a.out

Chooses Makefile.tau-icpc-mpi,pdt and associated libraries.

% tau_exec -T serial,pdt ./a.out
Chooses Makefile.tau-pdt or the shortest Makefile name without -mpi.
```

-T <list_of_tags> is used in several TAU tools:

- tau_run
- tau_rewrite
- tau_exec
- tau_gen_wrapper

Profiling Python applications

- Create a top-level Python wrapper
- Launch with tau_exec -T python …

```
% cat wrapper.py

#!/bin/env python
import tau

def OurMain():
    try:
        import <your_application>
except SystemExit:
        # Intercept the exit call so Tau writes profiles
        pass

tau.run('OurMain()')

% tau_exec -T python wrapper.py
```

Profiling Python applications

TAU: ParaProf: n,c,t 0,0,0 - flat.ppk

Metric: Time
Value: Exclusive
Units: seconds

5.473	dot
2.548	standard_normal
0.152	MachAr::__init__ [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/lib/machar.py} {50}]
0.112	_parse [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {385}]
0.105	_compile [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_compile.py} {38}]
0.056	any [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/fromnumeric.py} {540}]
0.048	append
0.046	len
0.046	Tokenizer::__next [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {188}]
0.045	any
0.035	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/linalg/linalg.py} {7}]
0.033	SubPattern::getwidth [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {146}]
0.033	_optimize_charset [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_compile.py} {213}]
0.028	Tokenizer::get [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {207}]
0.023	_parse_sub [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {307}]
0.022	add_newdoc [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/lib/function_base.py} {1154}]
0.019	abs
0.019	_compile_charset [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_compile.py} {184}]
0.016	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/__init__.py} {2}]
0.015	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/numeric.py} {1}]
0.014	SubPattern::__len__ [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {132}]
0.011	_mk_bitmap [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_compile.py} {264}]
0.011	Tokenizer::match [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {201}]
0.01	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/defmatrix.py} {1}]
0.009	SubPattern::__getitem__ [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {136}]
0.009	_simple [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_compile.py} {360}]
0.008	SubPattern::append [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {144}]
0.007	min
0.007	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/ctypes/__init__.py} {4}]
0.007	extend_all [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/numeric.py} {83}]
0.006	ord
0.005	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/__init__.py} {17}]
0.005	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/ma.py} {9}]
0.005	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/numerictypes.py} {76}]
0.005	SubPattern::__getslice__ [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {140}]
0.004	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/lib/index_tricks.py} {3}]
0.004	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/lib/__init__.py} {1}]
0.004	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/random/__init__.py} {2}]
0.004	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/opcode.py} {5}]
0.004	_extendLine [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/arrayprint.py} {270}]
0.004	_formatArray [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/arrayprint.py} {278}]

Instrumentation: Re-writing Binaries

Support for both static and dynamic executables

Specify the list of routines to instrument/exclude from instrumentation

Specify the TAU measurement library to be injected

Simplify the usage of TAU:

- To instrument:

```
% tau_run -T [tags] a.out -o a.inst
```

- To perform measurements, execute the application:

```
% mpiexec./a.inst
```

- To analyze the data:

```
% paraprof
```

Instrumentation: Re-writing Binaries with MAQAO

Simplify the usage of TAU:

- To instrument:

```
% tau_rewrite -T [tags] a.out -o a.inst
```

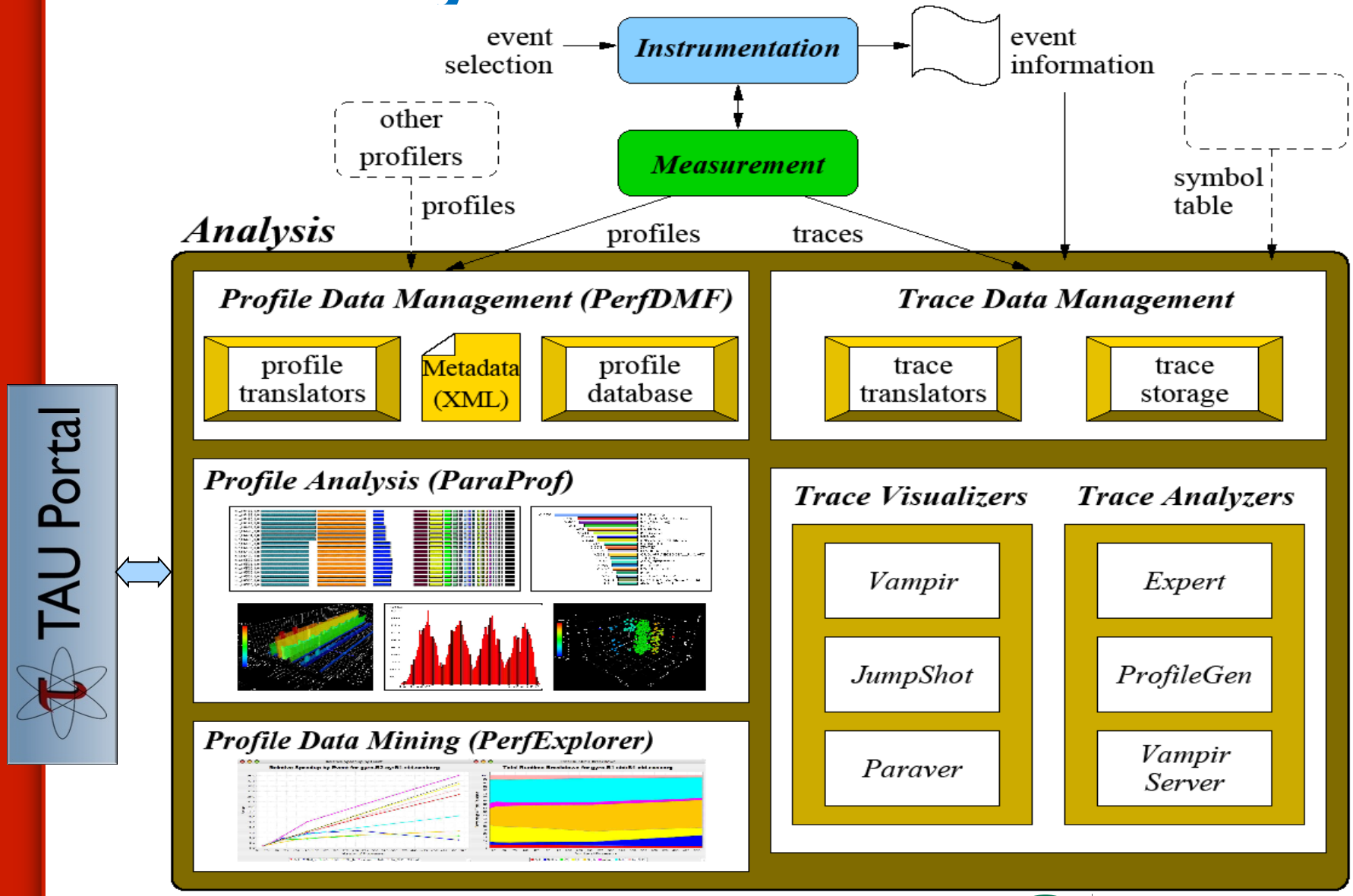
- To perform measurements, execute the application:

```
% mpiexec ./a.inst
```

- To analyze the data:

```
% paraprof
```

TAU Analysis



Performance Analysis

Analysis of parallel profile and trace measurement

Parallel profile analysis (ParaProf)

- Java-based analysis and visualization tool
- Support for large-scale parallel profiles

Performance data management framework (PerfDMF)

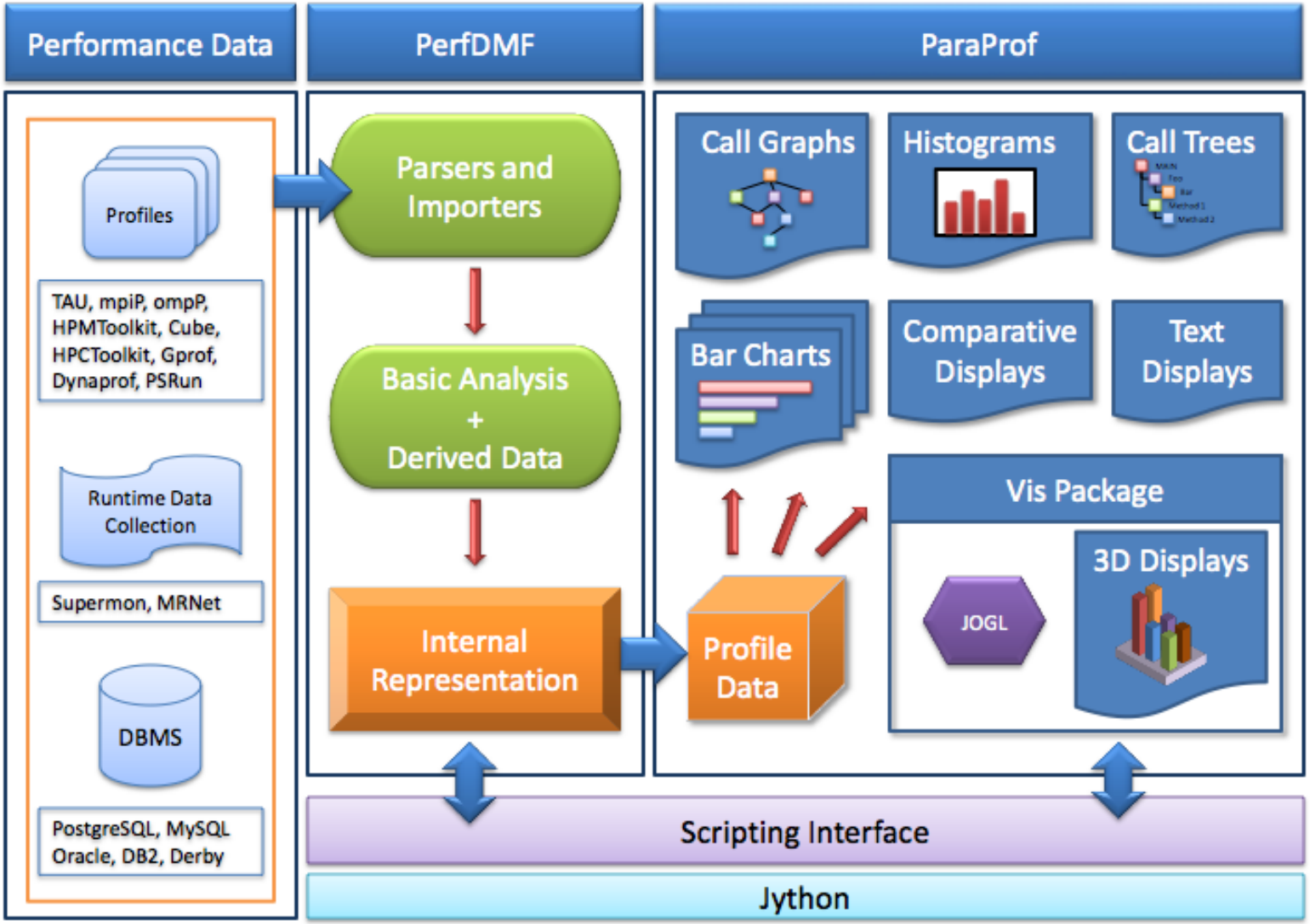
Parallel trace analysis

- Translation to ParaVer, SLOG-2, OTF formats
- Integration with Vampir / Vampir Pro (TU Dresden)
- Profile generation from trace data

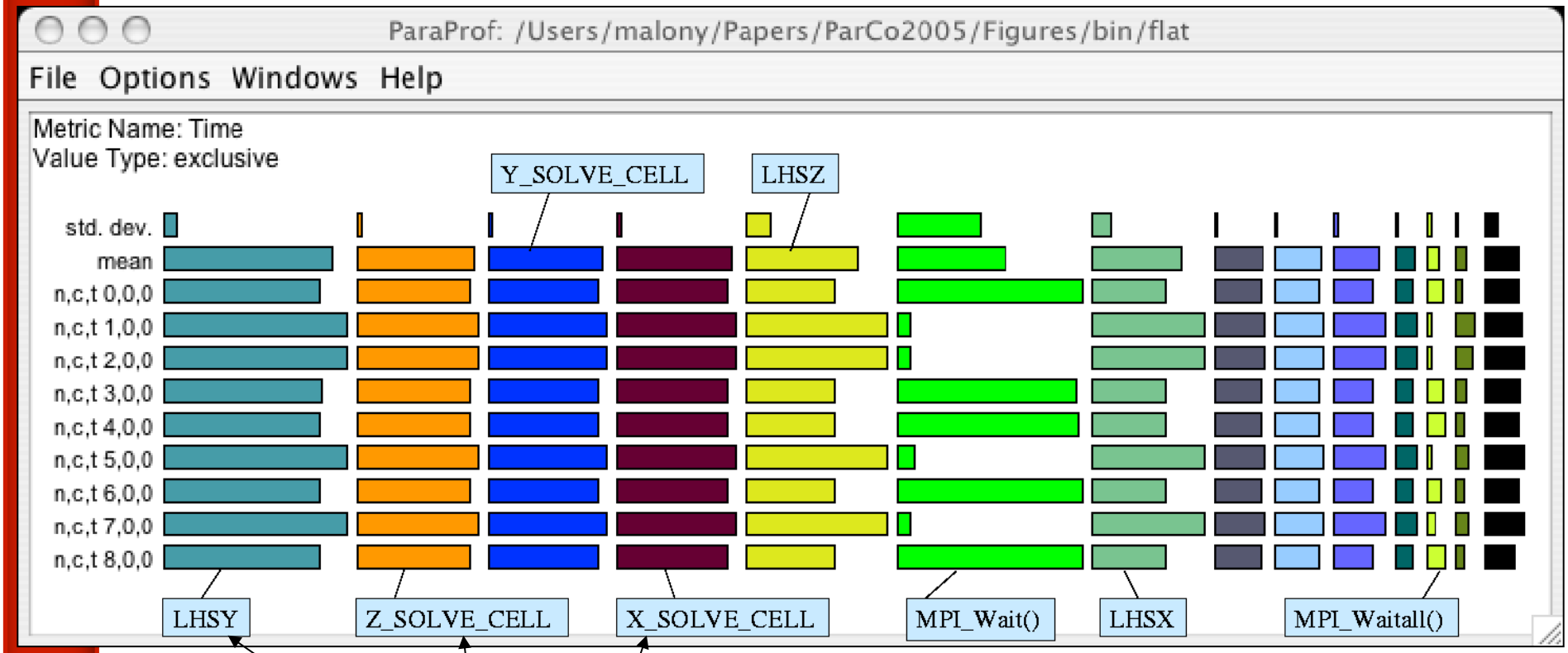
Online parallel analysis and visualization

Integration with CUBE browser (Scalasca, FZJ)

ParaProf Profile Analysis Framework



NAS BT – Flat Profile



Application routine names reflect phase semantics

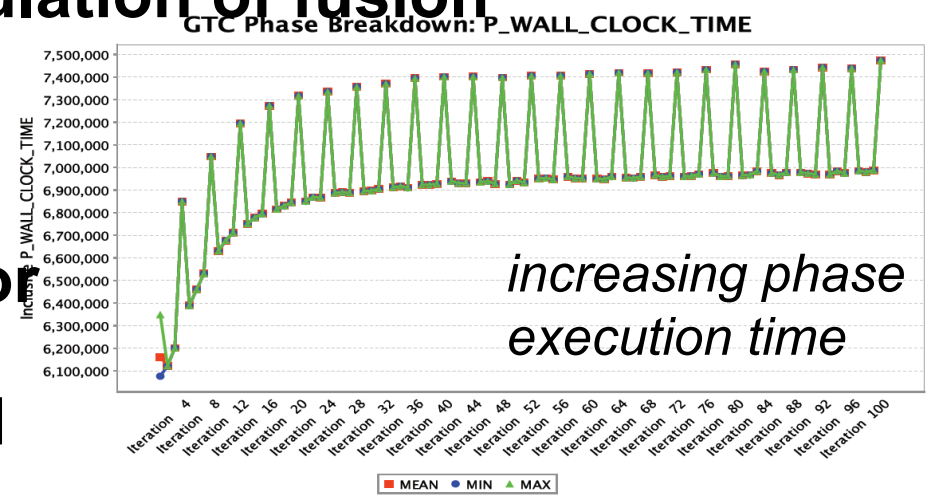
How is MPI_Wait() distributed relative to solver direction?

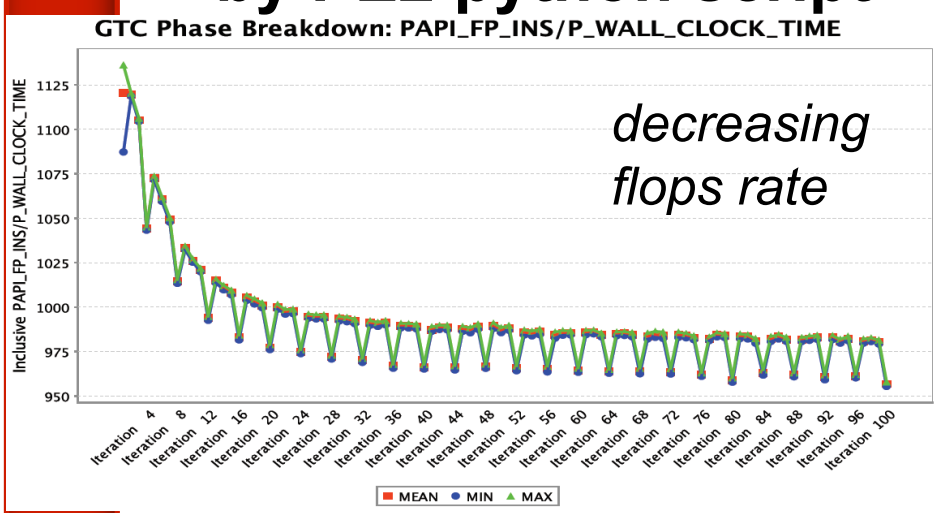
NAS BT – Phase Profile

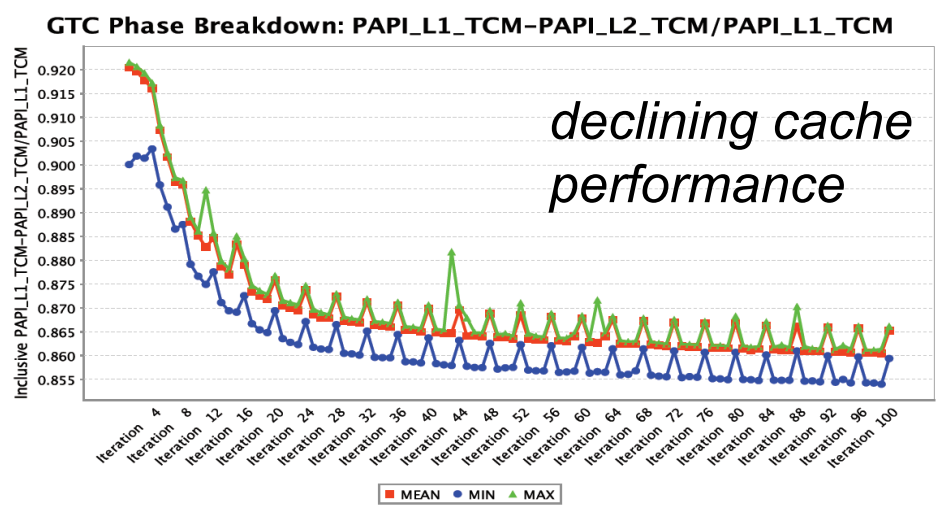
Main phase shows nested phases and immediate events

Phase Profiling of HW Counters

- GTC particle-in-cell simulation of fusion turbulence
- Phases assigned to iterations
- Poor temporal locality for one important data
- Automatically generated by PE2 python script



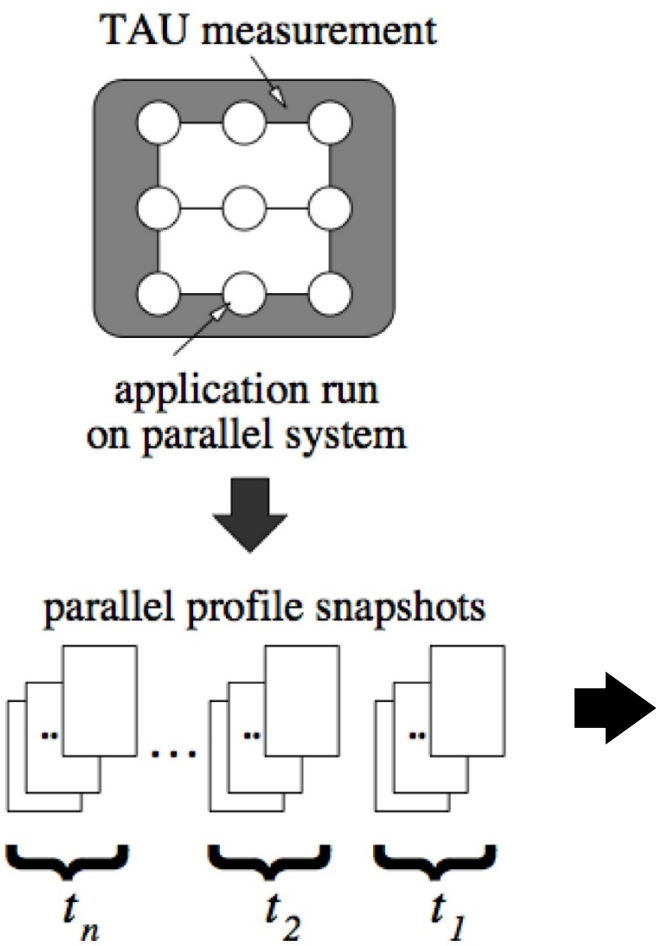


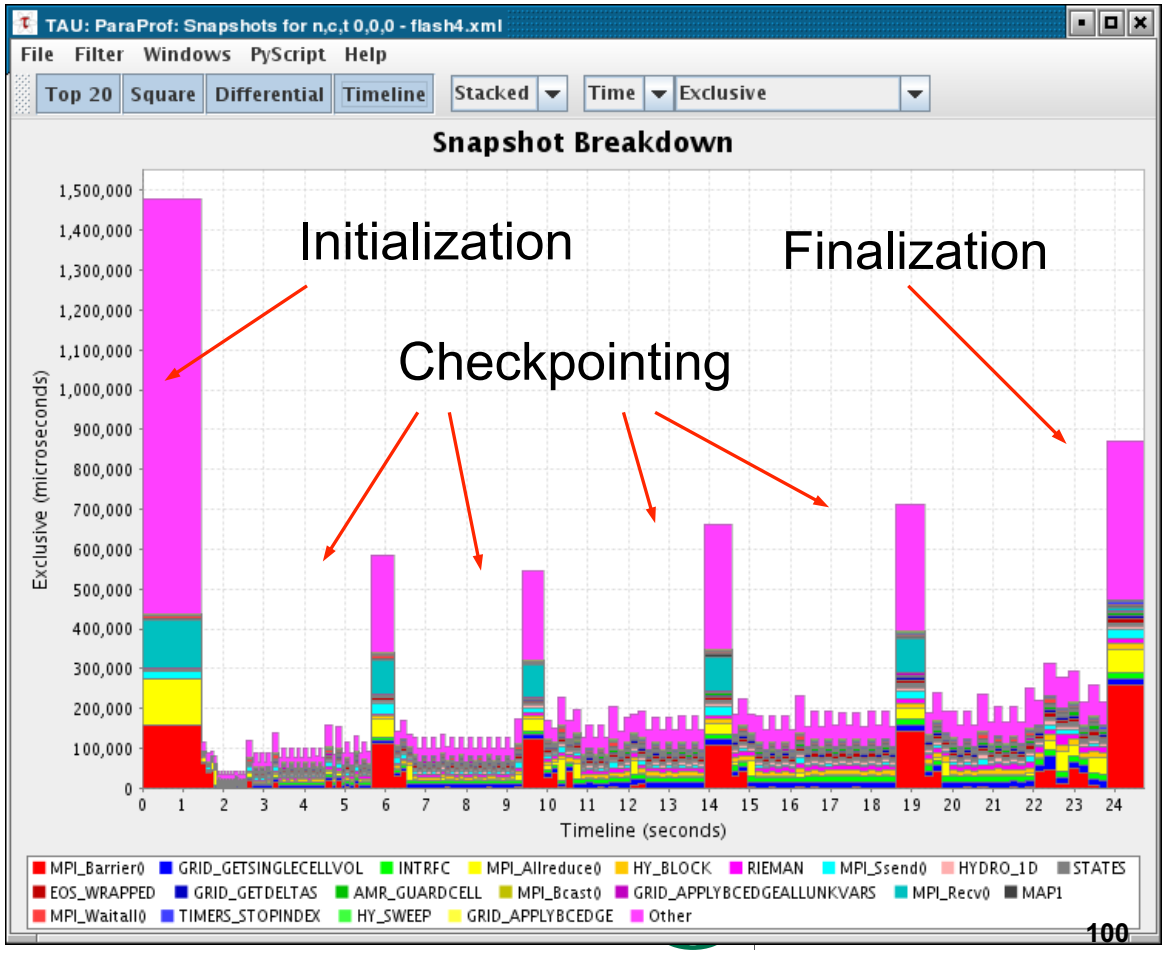


Profile Snapshots in ParaProf

Profile snapshots are parallel profiles recorded at runtime

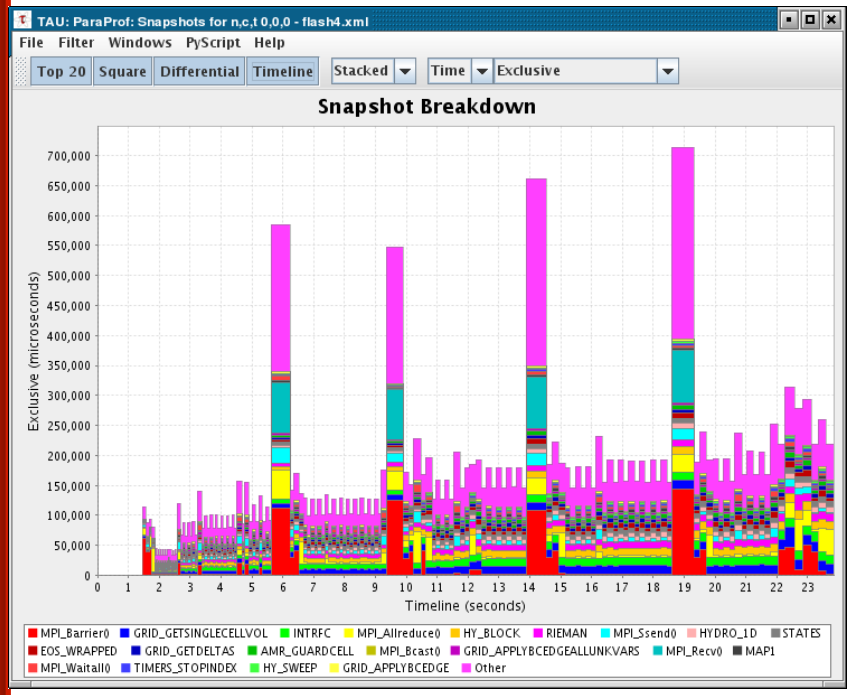
Shows performance profile dynamics (all types allowed)



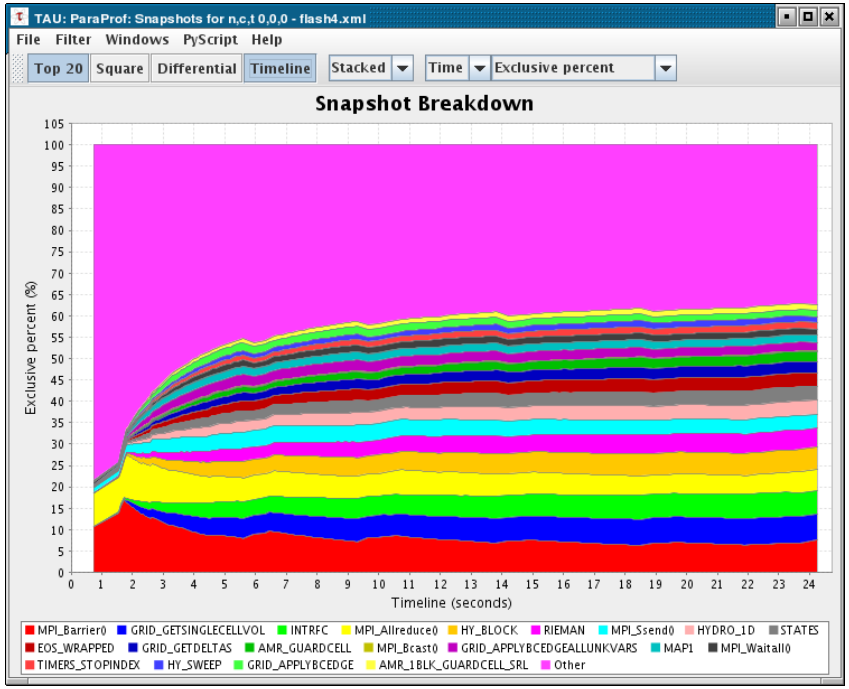


Profile Snapshot Views

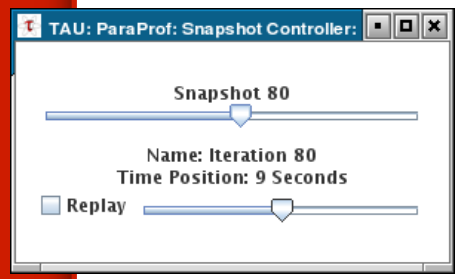
Percentage breakdown



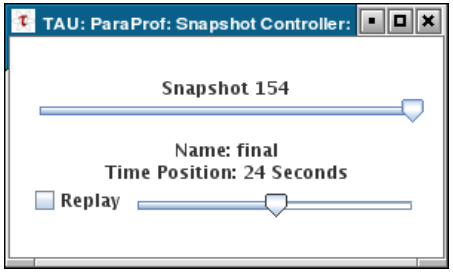
Only show main loop

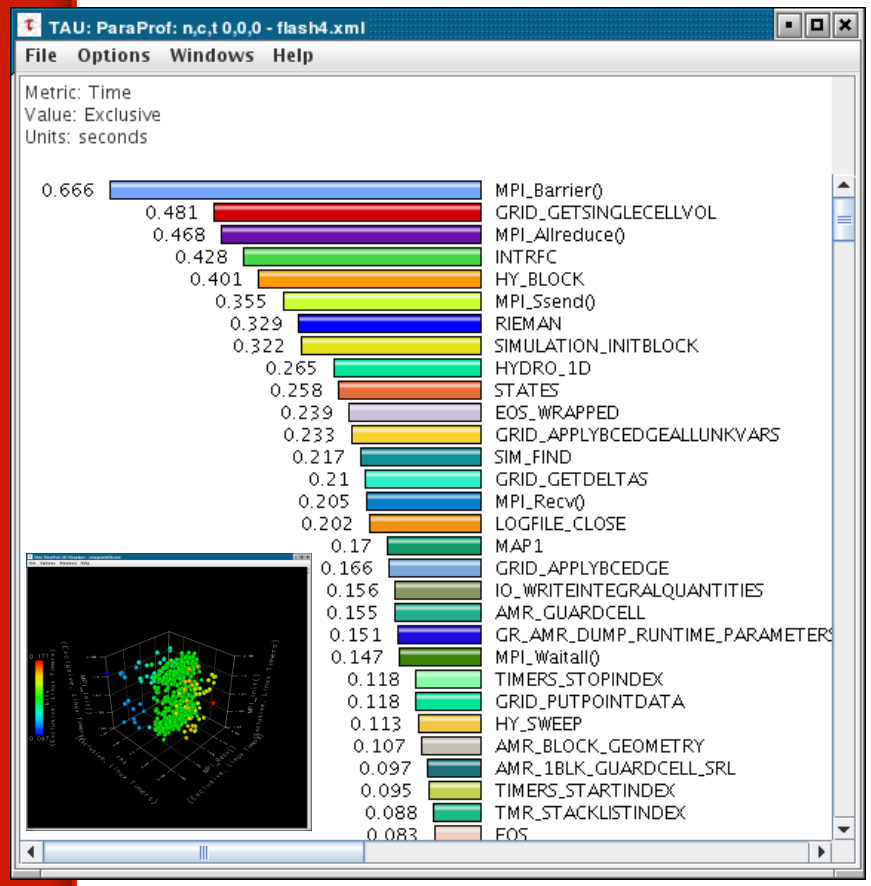


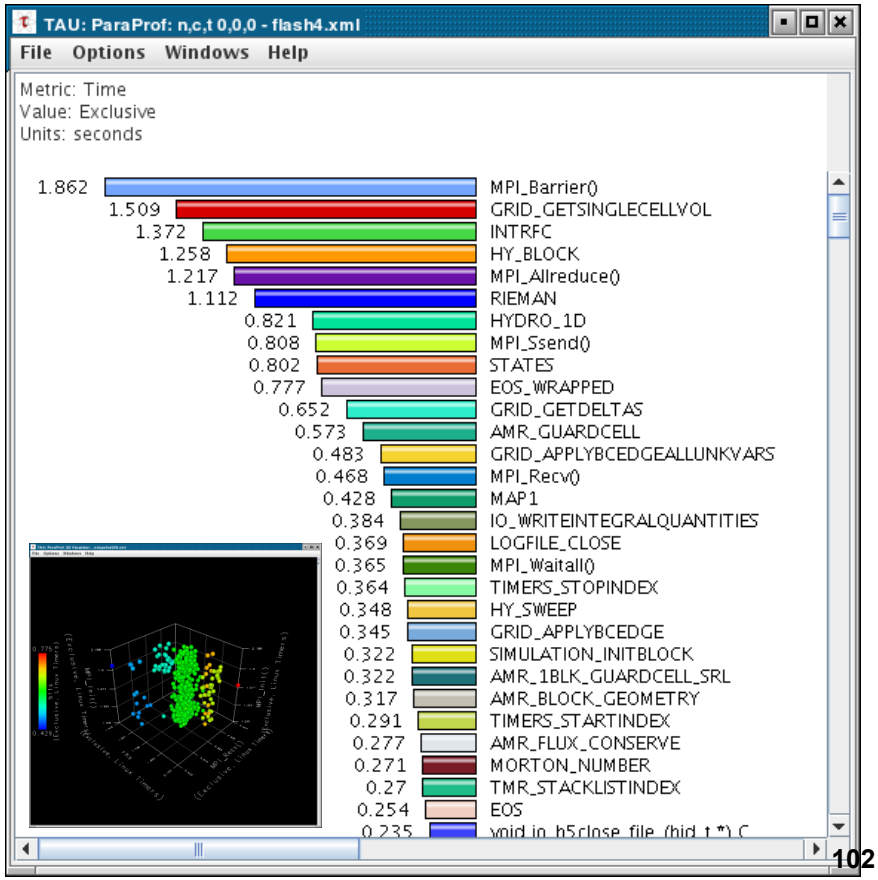
Snapshot Replay in ParaProf



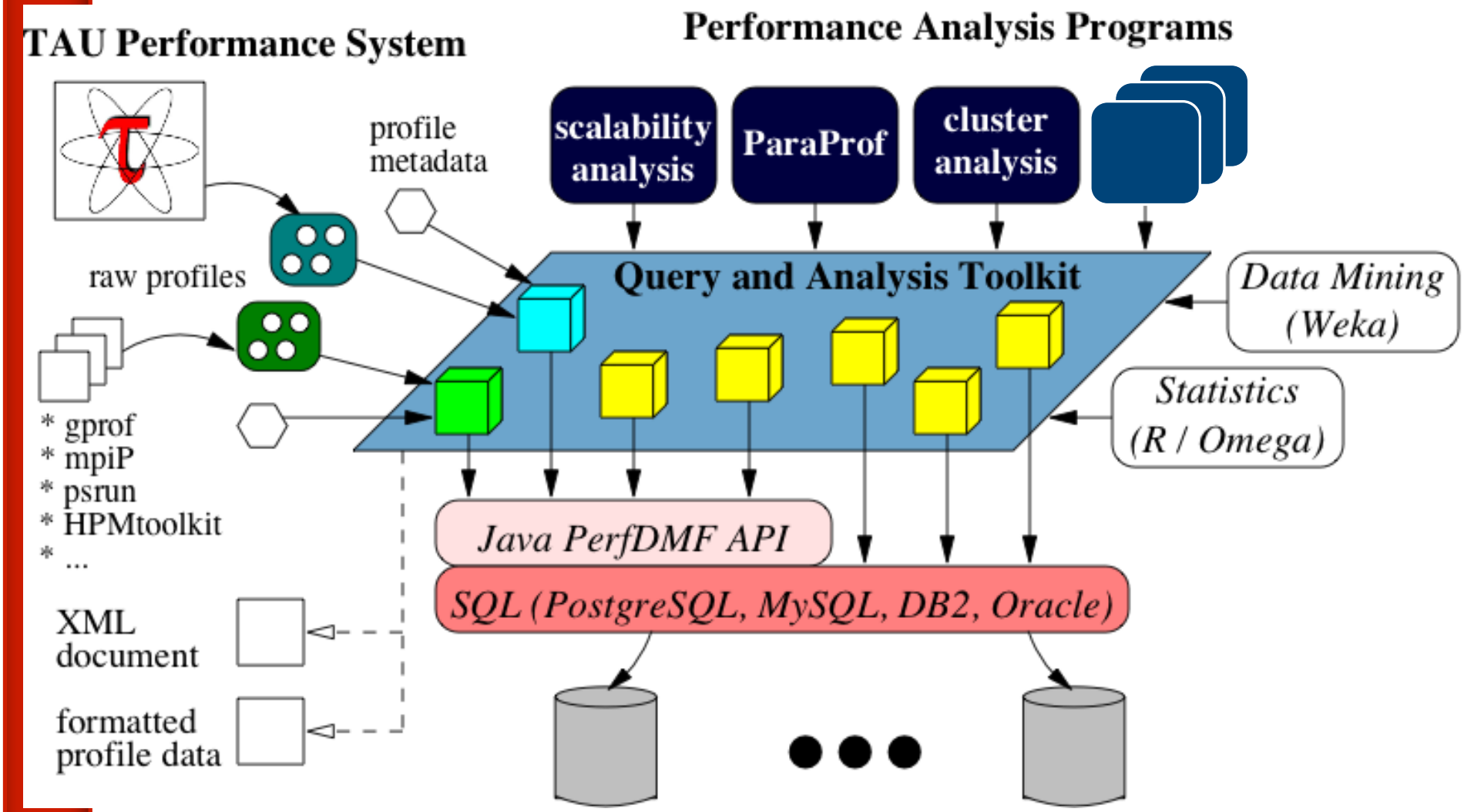
All windows dynamically update



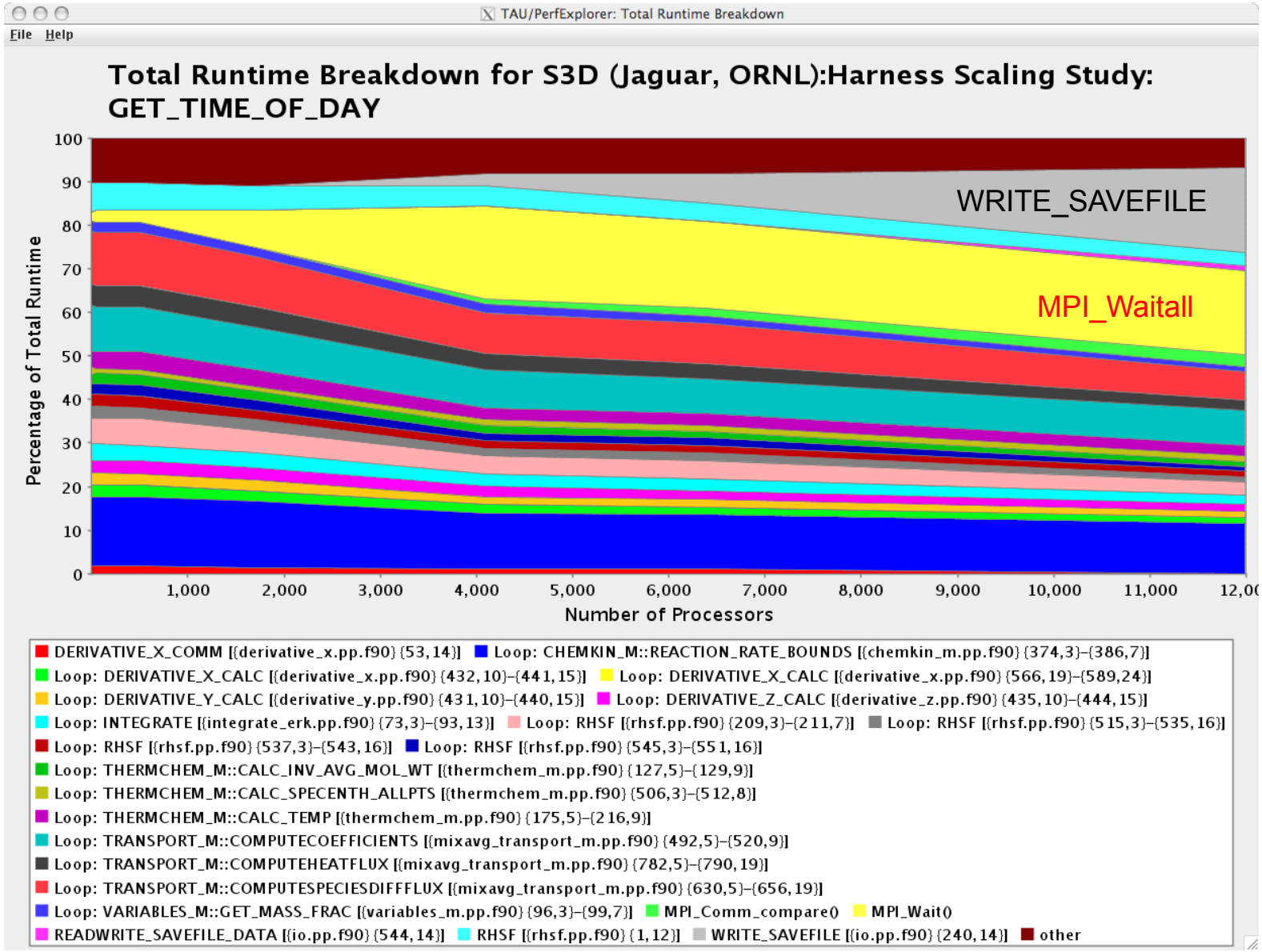




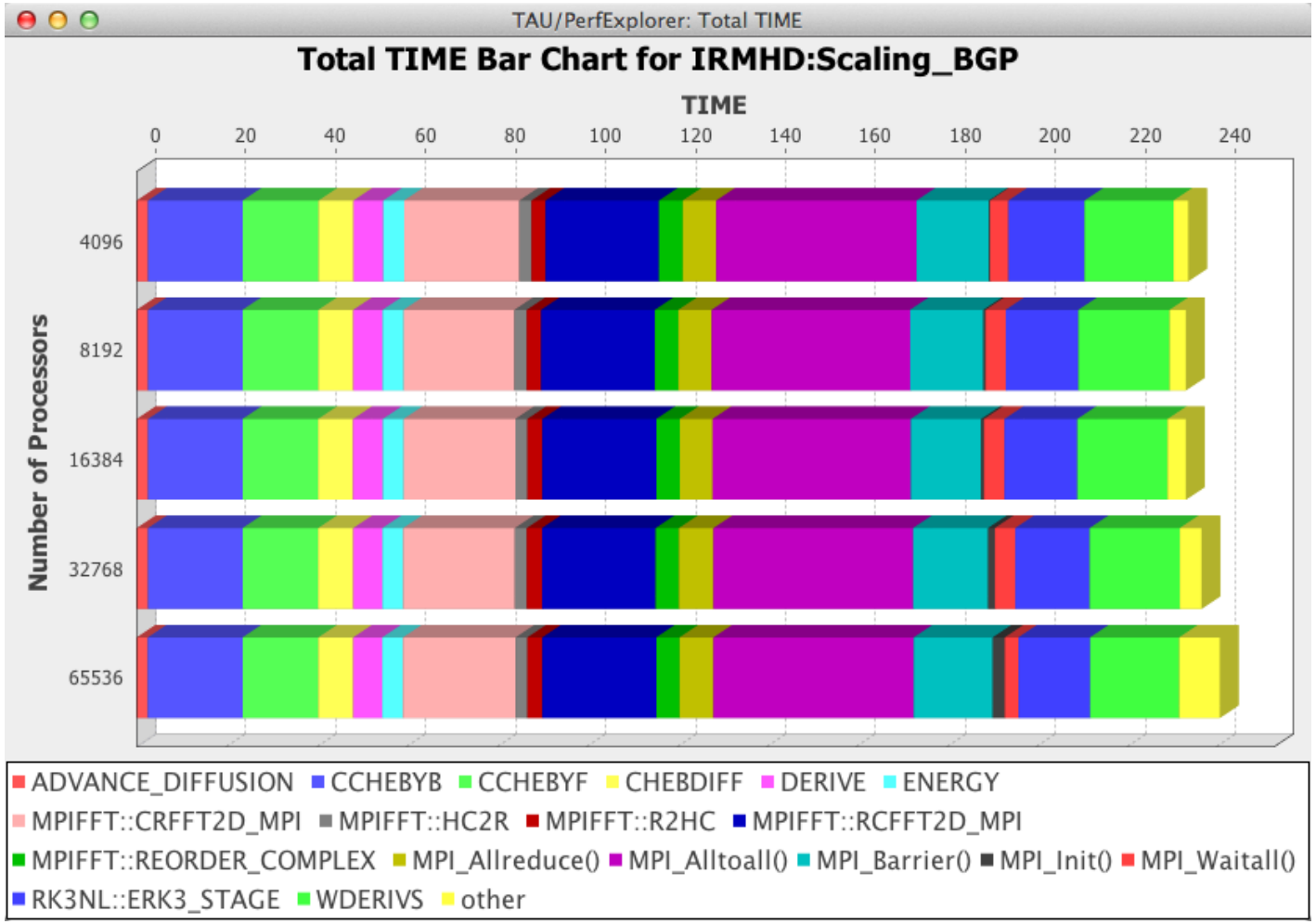
TAUdb: Framework for Managing Performance Data



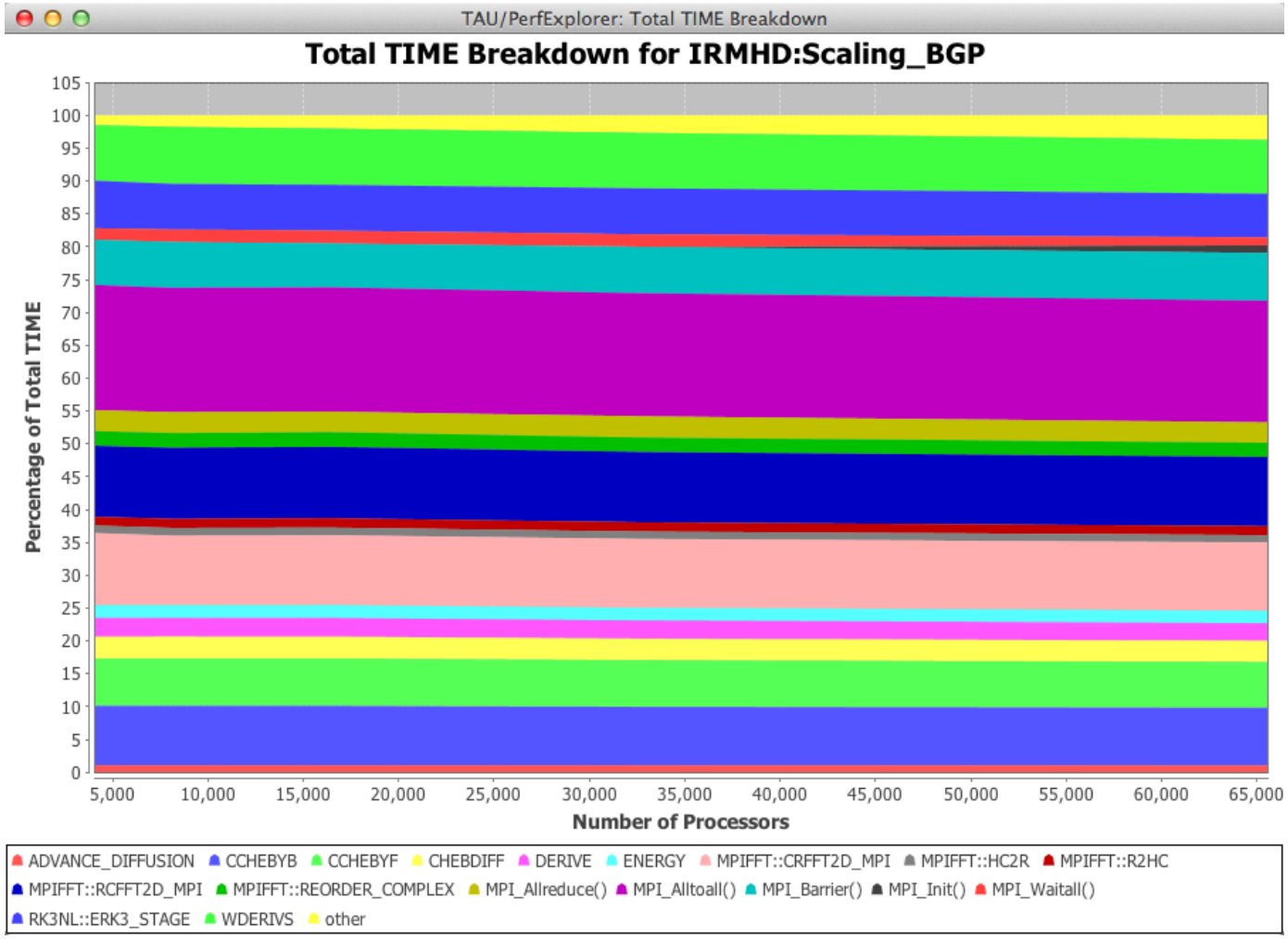
PerfExplorer – Runtime Breakdown



Evaluate Scalability

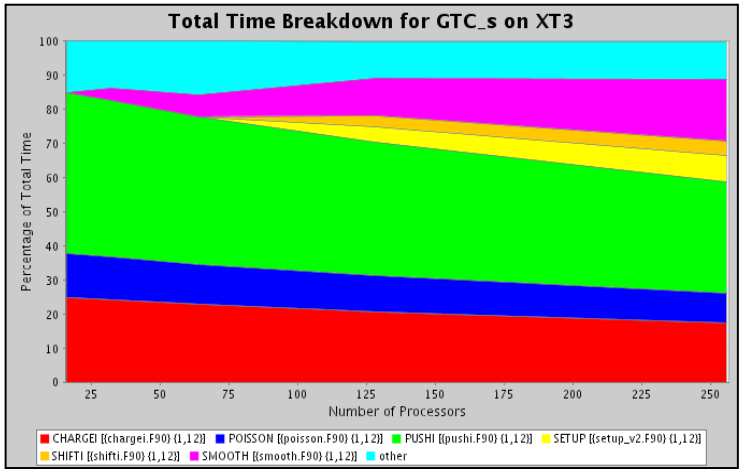


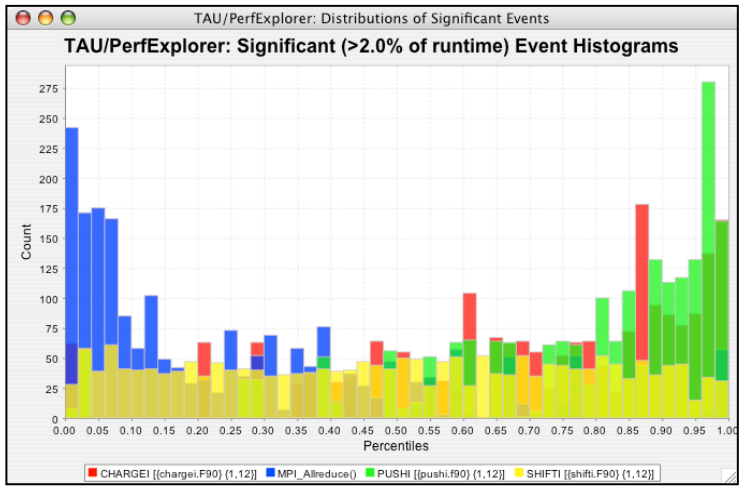
Runtime Breakdown



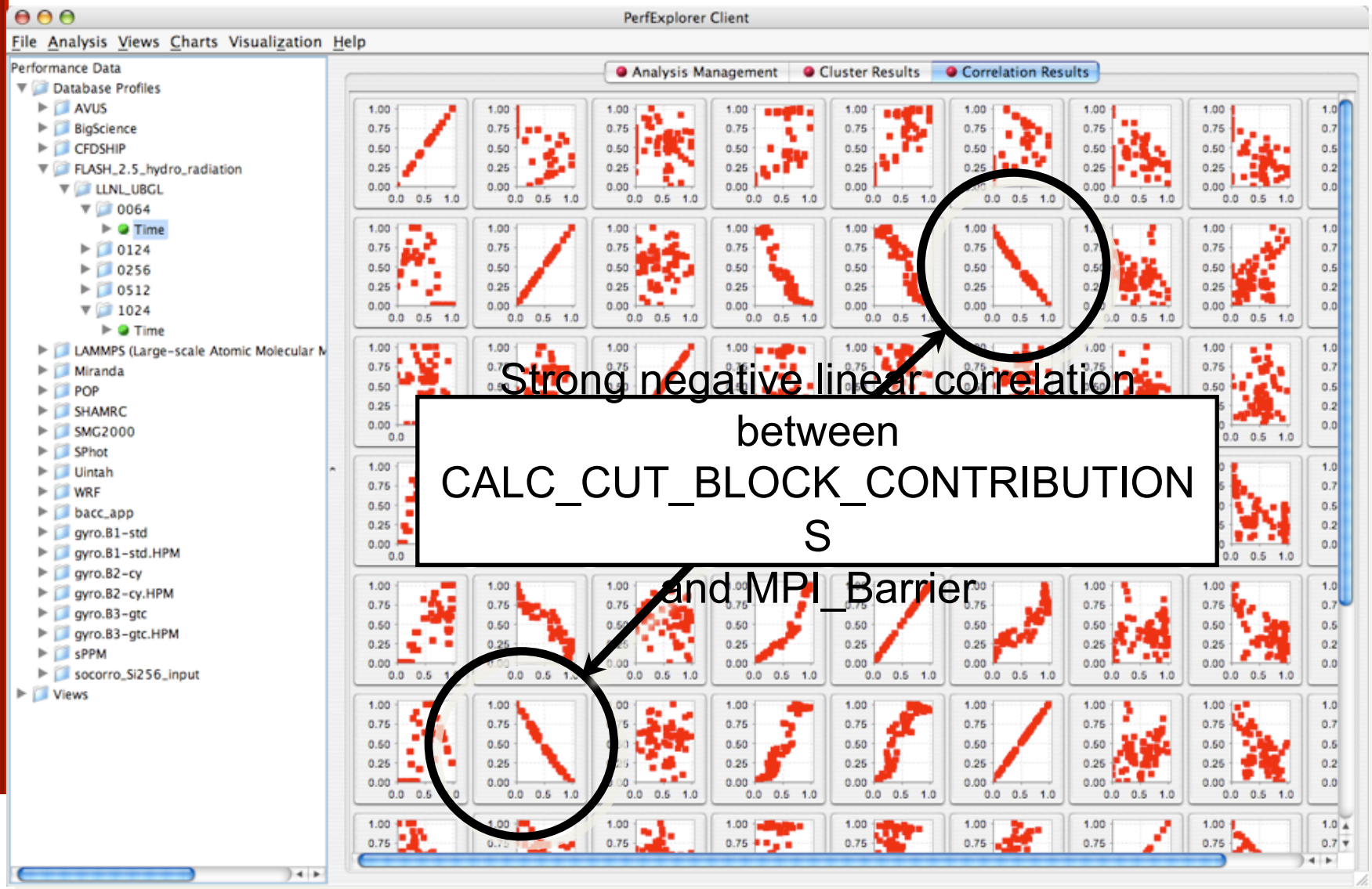
PerfExplorer – Relative Comparisons

- Total execution time
- Timesteps per second
- Relative efficiency
- Relative efficiency per event
- Relative speedup
- Relative speedup per event
- Group fraction of total
- Runtime breakdown
- Correlate events with total runtime
- Relative efficiency per phase
- Relative speedup per phase
- Distribution visualizations



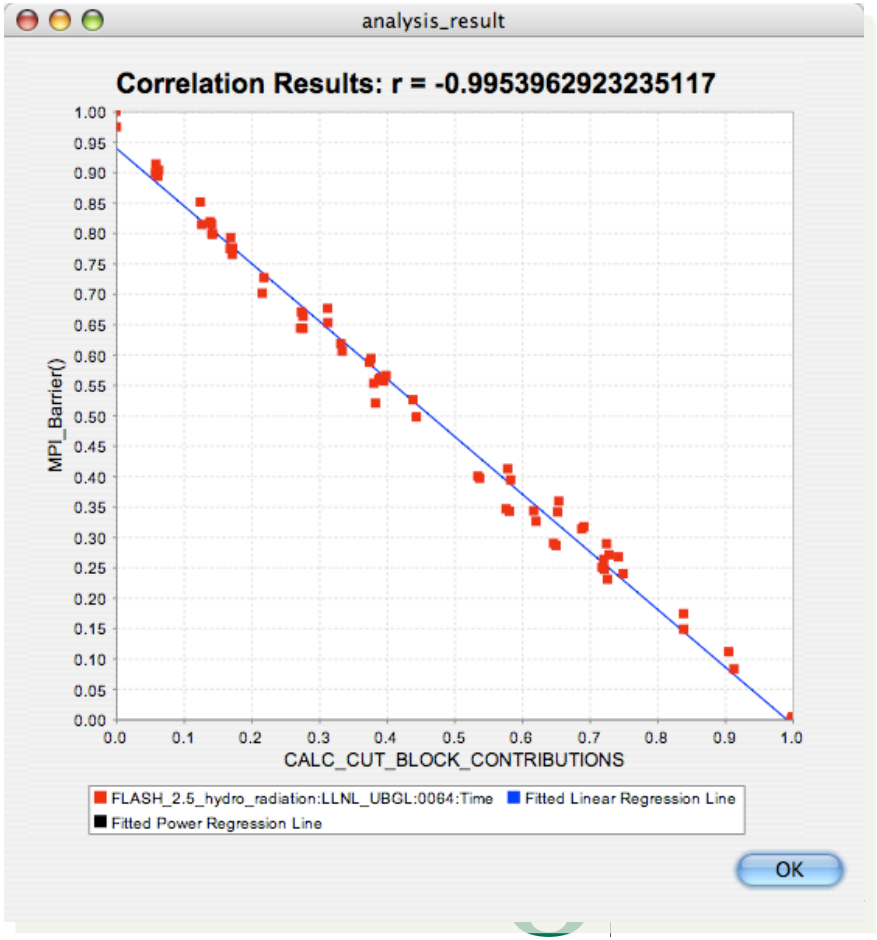


PerfExplorer – Correlation Analysis

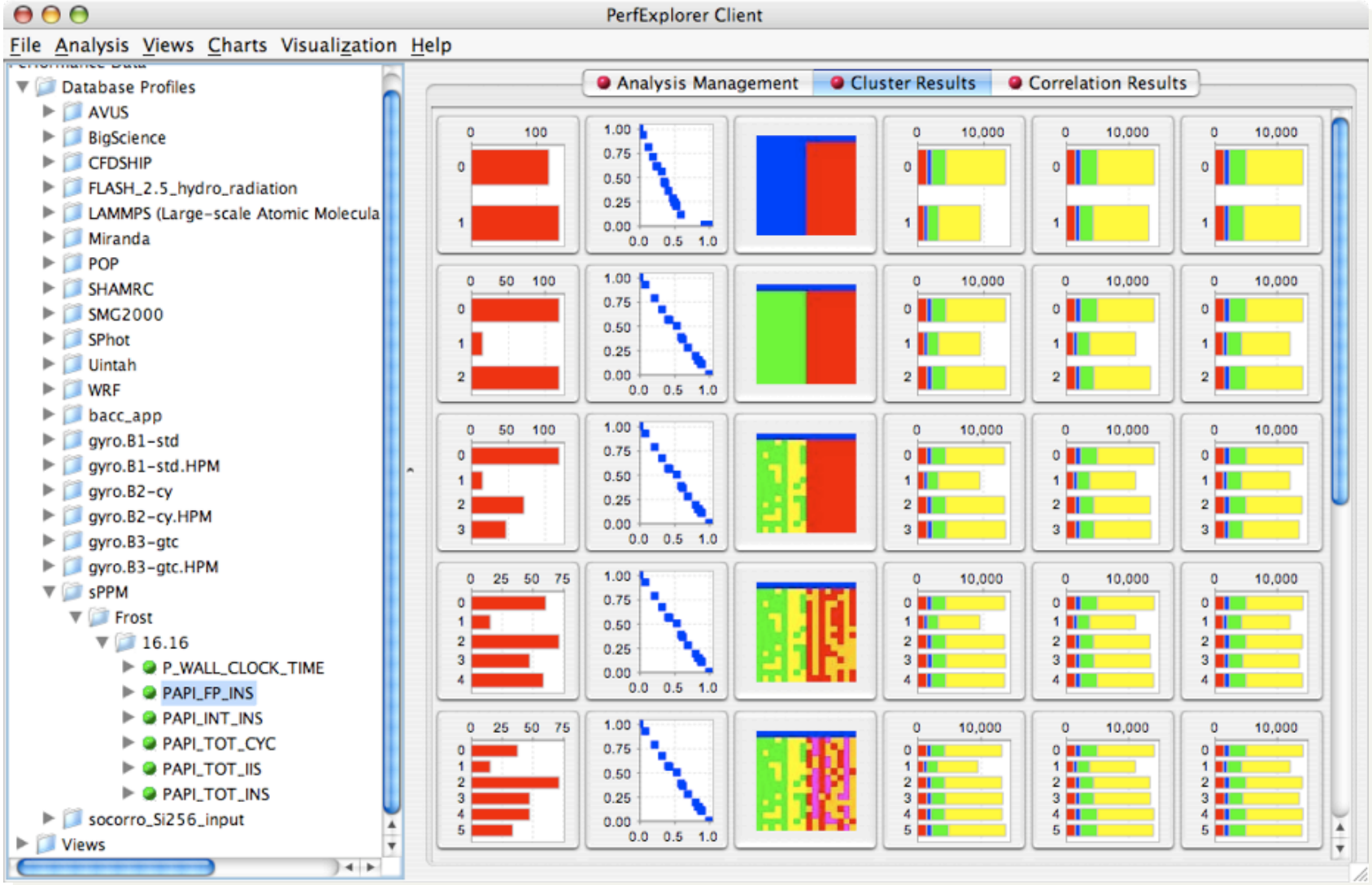


PerfExplorer – Correlation Analysis

-0.995 indicates strong, negative relationship. As CALC_CUT_BLOCK_CONTRIBUTIONS() increases in execution time, MPI_Barrier() decreases



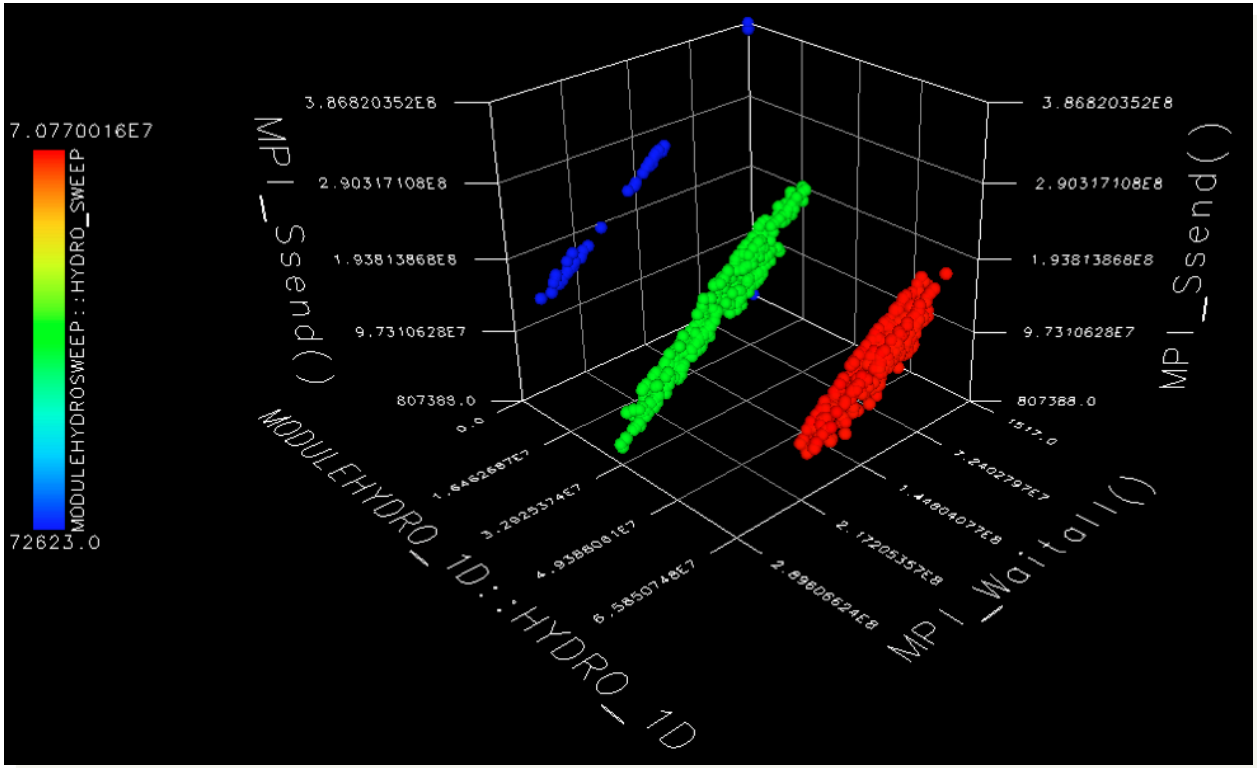
PerfExplorer – Cluster Analysis



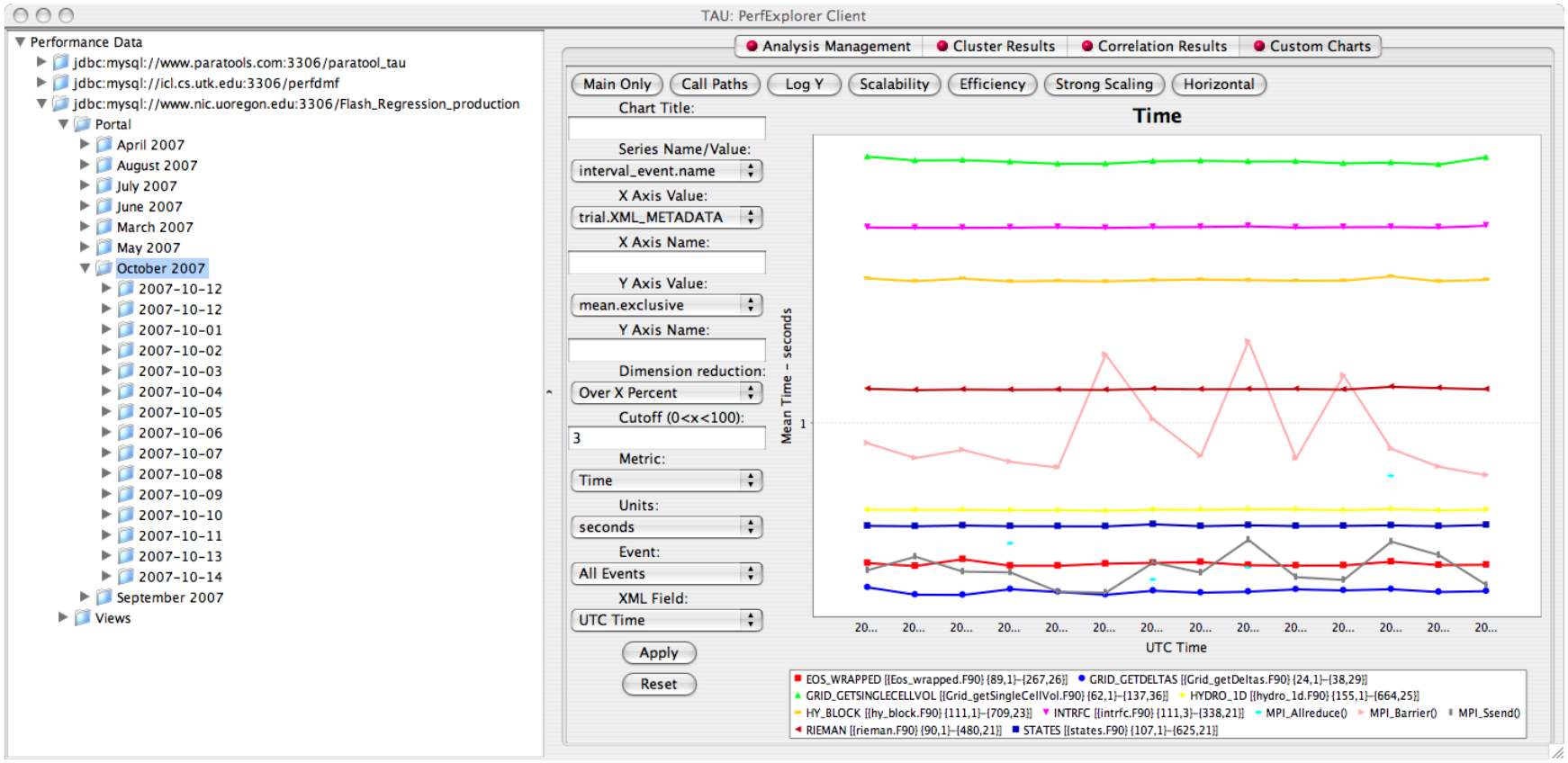
PerfExplorer – Cluster Analysis

Four significant events automatically selected

Clusters and correlations are visible



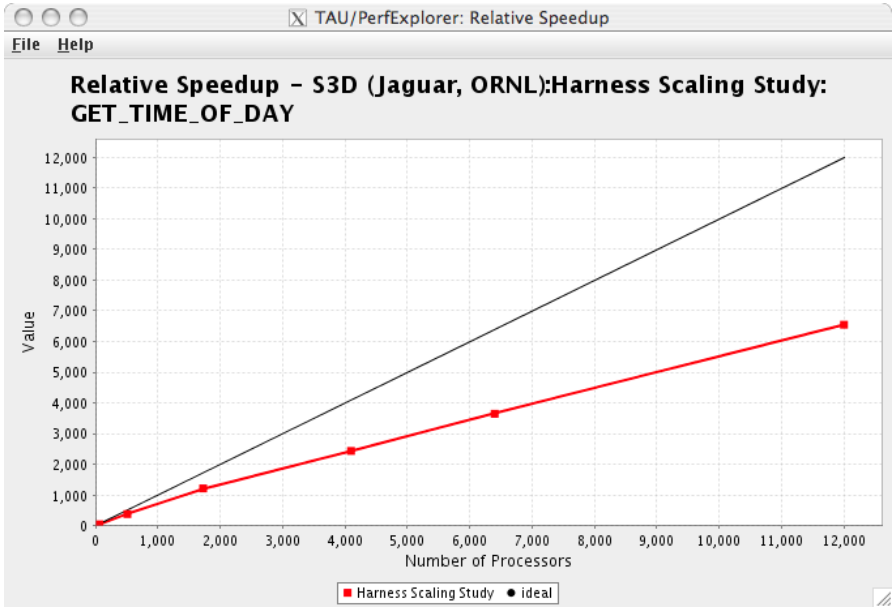
PerfExplorer: Performance Regression

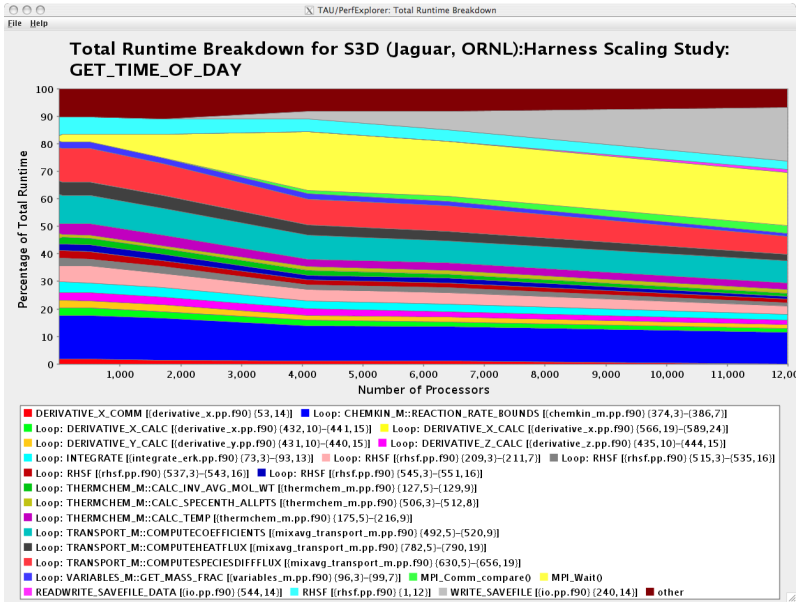


Evaluate Scalability

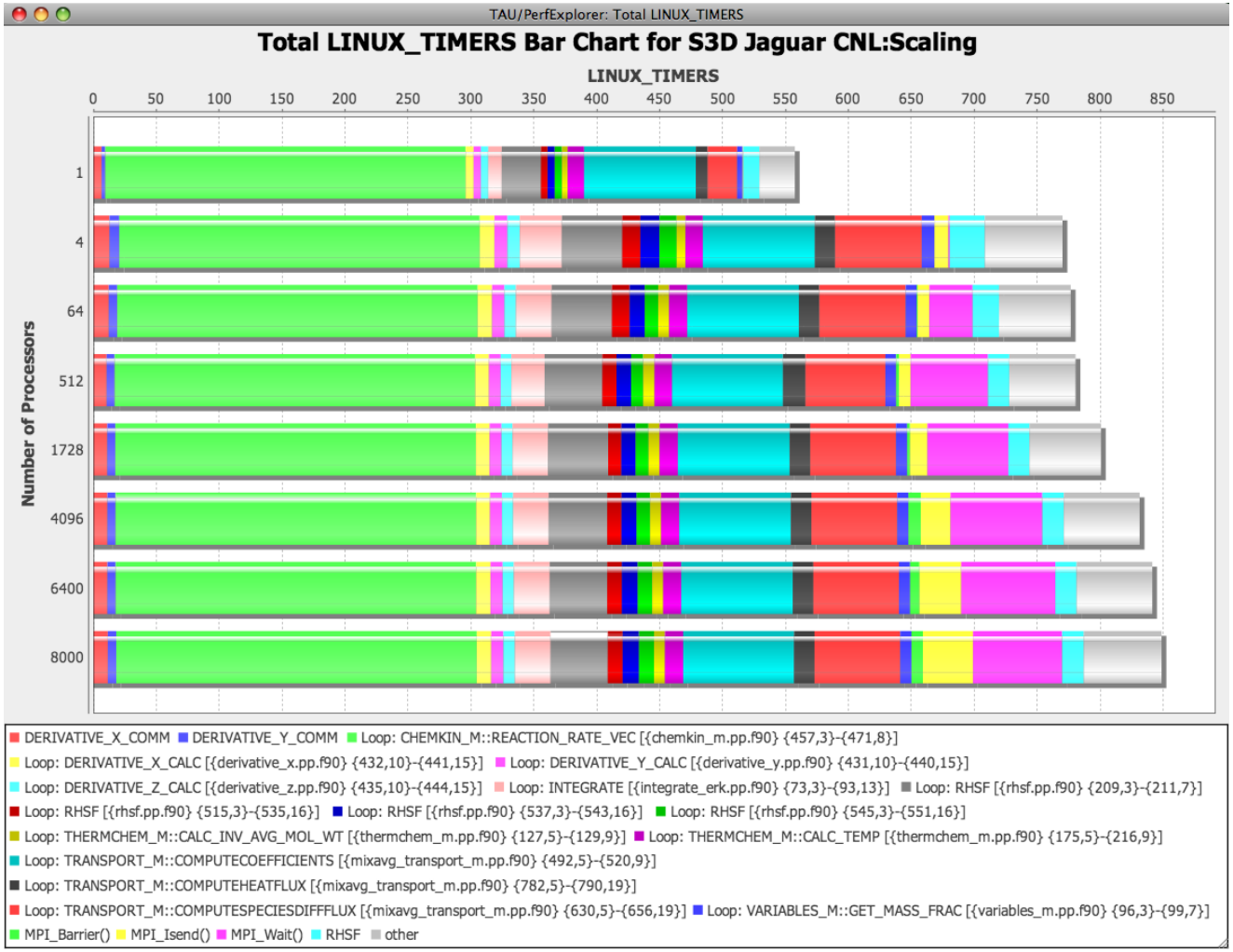
Goal: How does my application scale? What bottlenecks at what CPU counts?

Load profiles in PerfDMF database and examine with PerfExplorer

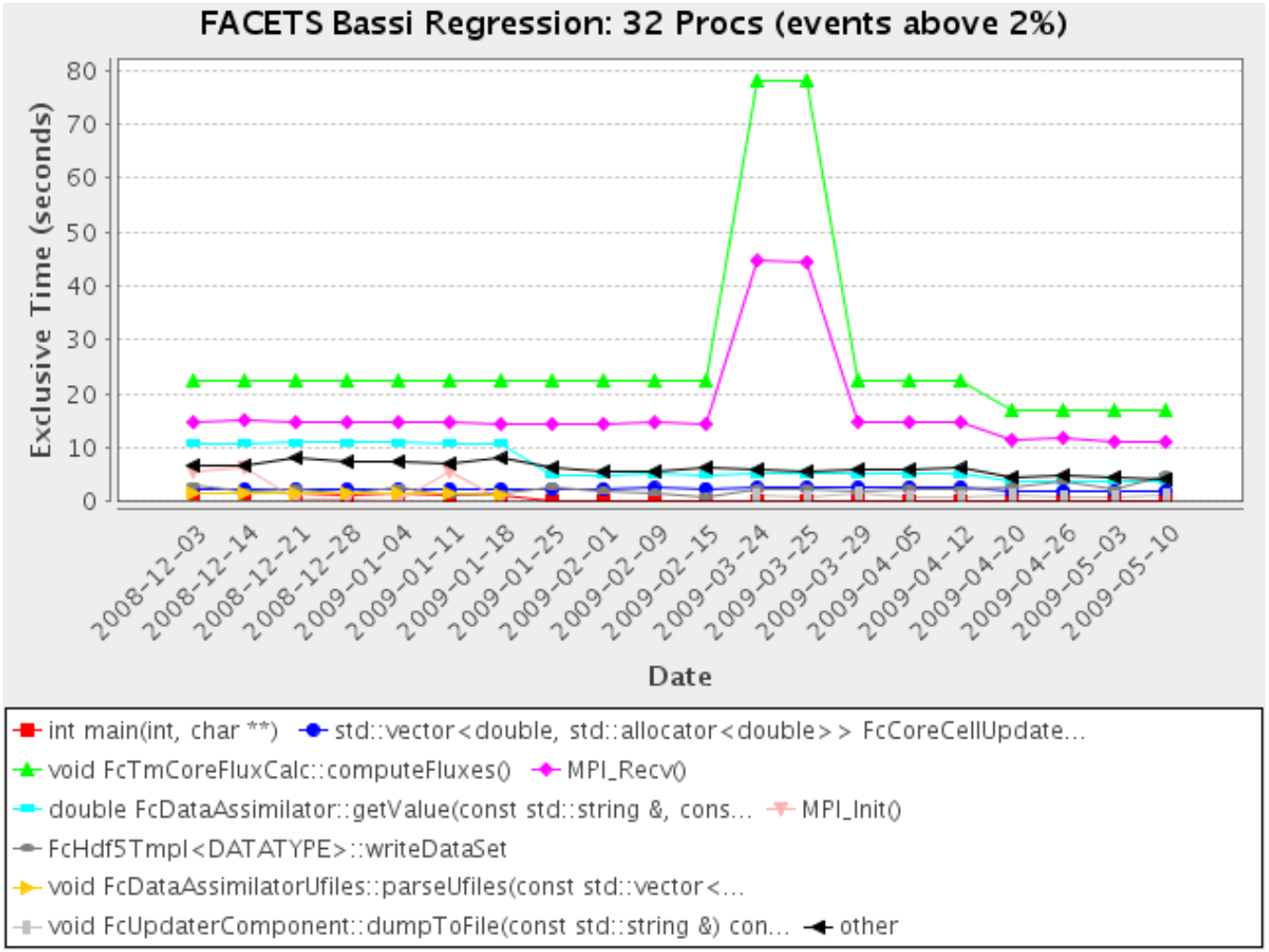




Usage Scenarios: Evaluate Scalability



Performance Regression Testing



Evaluate Scalability using PerfExplorer Charts

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% export PATH=$TAU/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% qsub  run1p.job
% paraprof --pack 1p.ppk
% qsub  run2p.job …
% paraprof --pack 2p.ppk … and so on.
On your client:
% taudb_configure -create-default
% perfexplorer_configure
(Enter, Yes to load jars, schema, defaults)
% paraprof
(load each trial: DB -> Add Trial -> Type (Paraprof Packed
Profile) -> OK, OR use taudb_loadtrial on the commandline)
% taudb_loadtrial -a Matmult -x Scaling -n 32p 32p.ppk
% perfexplorer
(Charts -> Speedup)
```

Multi-language Application Debugging

```
% qsub -I …

% mpirun -np 4 tau_exec -memory_debug ./a.out
% paraprof

Other options:
% export TAU_MEMDBG_PROTECT_ABOVE=1
% export TAU_MEMDBG_PROTECT_BELOW=1
% export TAU_MEMDBG_PROTECT_FREE=1
```

Multi-language Application Debugging

TAU: ParaProf Manager

- Applications
  - Standard Applications
    - Default App
      - Default Exp
        - debug.ppk
          - TIME
  - Default (jdbc:h2:/User:
  - alcf (jdbc:postgresql:/,
  - perfexplorer_working (
  - taudb (jdbc:postgresql

TrialField	Value
Name	debug.ppk
Application ID	0
Experiment ID	0
Trial ID	0
BACKTRACE(1) 1	[bar] [/glade/u/home/sshende/pkgs/workshop/debug/foo.c:17] [/glade/u/home/sshende/pkgs/workshop/debug/foo.tau_rewritten]
BACKTRACE(1) 2	[foo] [/glade/u/home/sshende/pkgs/workshop/debug/foo.c:24] [/glade/u/home/sshende/pkgs Show Source Code written]
BACKTRACE(1) 3	[main] [/glade/u/home/sshende/pkgs/workshop/debug/foo.c:31] [/glade/u/home/sshende/pkgs/workshop/debug/foo.tau_rewritten]
BACKTRACE(1) 4	[__libc_start_main] [(unknown):0] [/lib64/libc-2.12.so]
BACKTRACE(1) 5	[_start] [(unknown):0] [/glade/u/home/sshende/pkgs/workshop/debug/foo.tau_rewritten]
CPU Cores	8
CPU MHz	2600.000
CPU Type	Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz
CPU Vendor	GenuineIntel
CWD	/glade/u/home/sshende/pkgs/workshop/debug
Cache Size	20480 KB
Command Line	/glade/u/home/sshende/pkgs/workshop/debug/foo.tau_rewritten
Executable	/glade/u/home/sshende/pkgs/workshop/debug/foo.tau_rewritten
File Type Index	0
File Type Name	ParaProf Packed Profile
Hostname	ys0101
Local Time	2013-03-31T17:13:40-06:00
MPI Processor Name	ys0101
Memory Size	32988340 kB
Node Name	ys0101
OS Machine	x86_64
OS Name	Linux
OS Release	2.6.32-220.13.1.el6.x86_64
OS Version	#1 SMP Thu Mar 29 11:46:40 EDT 2012
SIGNAL	Segmentation fault
Starting Timestamp	1364771620575991
TAU Architecture	x86_64
TAU Config	-iowrapper -mpiinc=/opt/ibmhpc/pe1209/mpich2/intel/include64 -mpilib=/opt/ibmhpc/pe1209/mpich2/intel/lib64 -cc=icc -c++...
TAU Makefile	/glade/u/home/sshende/pkgs/tau-2.22.2/x86_64/lib/Makefile.tau-icpc-mpi-pdt
TAU Version	2.22.2

Location of segmentation violation

TAU: ParaProf: Source Browser: /glade/u/home/sshe...

File Help

```
1   #include <stdio.h>
2   #include <mpi.h>
3   #include <unistd.h>
4   #include <stdlib.h>
5
6   struct node {
7     int id;
8     struct node *next;
9   };
10
11  int bar(int x) {
12    int y;
13    struct node *t = (struct node *)malloc(sizeof(struct node));
14    t->next = NULL;
15    t->id = x;
16    printf("t -> id = %d\n", t->id);
17    y = t->next->id;
18    printf("y=%d\n",y);
19    return x;
20  }
21
22  int foo(int x) {
23    printf("foo: x = %d\n", x);
24    bar(x);
25    return x;
26  }
27
28  int main(int argc, char **argv) {
29    int ret;
30    MPI_Init(&argc, &argv);
31    ret = foo(29);
32    MPI_Finalize();
33    return ret;
34  }
```

Memory Leak Detection

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% export PATH=$TAU/../bin:$PATH
% export TAU_OPTIONS='-optMemDbg -optVerbose'
% make F90=tau_f90.sh CC=tau_cc.sh CXX=tau_cxx.sh
% export TAU_TRACK_MEMORY_LEAKS=1
% mpiexec ./matmult
% paraprof
```

Multi-language Memory Leak Detection

TAU: ParaProf: Context Events for: node 0 – memleak.ppk

Name	Total	NumSamples	MaxValue	MinValue	MeanValue	Std. Dev.
Heap Allocate	5,000,033	2	5,000,001	32	2,500,016.5	2,499,984.5
Heap Allocate <file=simple.c, line=15>	180	3	80	48	60	14.236
Heap Allocate <file=simple.c, line=23>	180	1	180	180	180	0
Heap Free <file=simple.c, line=18>	80	1	80	80	80	0
Heap Free <file=simple.c, line=25>	180	1	180	180	180	0
Heap Memory Used (KB)	4,884.829	8	4,883.196	0.047	610.604	1,614.888
▼ int foo(int) C [{simple.c} {36,1}-{44,1}]						
▼ int bar(int) C [{simple.c} {7,1}-{28,1}]						
Heap Allocate <file=simple.c, line=23>	180	1	180	180	180	0
Heap Free <file=simple.c, line=25>	180	1	180	180	180	0
▼ int g(int) C [{simple.c} {30,1}-{34,1}]						
▼ int bar(int) C [{simple.c} {7,1}-{28,1}]						
Heap Allocate <file=simple.c, line=15>	180	3	80	48	60	14.236
Heap Free <file=simple.c, line=18>	80	1	80	80	80	0
MEMORY LEAK! Heap Allocate <file=simple.c, line=15>	100	2	52	48	50	2
▼ int main(int, char **) C [{simple.c} {45,1}-{55,1}]						
▼ MPI_Finalize()						
Heap Allocate	5,000,033	2	5,000,001	32	2,500,016.5	2,499,984.5
MEMORY LEAK! Heap Allocate	5,000,033	2	5,000,001	32	2,500,016.5	2,499,984.5

TAU Integration with IDEs

High performance software development environments

- Tools may be complicated to use
- Interfaces and mechanisms differ between platforms / OS

Integrated development environments

- Consistent development environment
- Numerous enhancements to development process
- Standard in industrial software development

Integrated performance analysis

- Tools limited to single platform or programming language
- Rarely compatible with 3rd party analysis tools
- Little or no support for parallel projects

TAU and Eclipse

Fortran – matmult.f90 – Eclipse SDK

File Edit Refactor Navigate Search Project Run Window Help

Fortran Projects | Navigator

- matmultiply
  - Binaries
  - Includes
  - .settings
  - Debug_ia32
  - Debug_ia32(tau-icpc-mpi-pdt)
  - matmult.f90
  - .cdtbuild
  - .cdtproject
  - .project
  - profile.0.0.0
  - profile.1.0.0
  - profile.2.0.0
  - profile.3.0.0

matmult.f90

```
!*********************************************************************
!     matmult.f90 - simple matrix multiply implementation
!*********************************************************************
      subroutine initialize(a, b, n)
        double precision a(n,n)
        double precision b(n,n)
        integer n

! first initialize the A matrix
        do i = 1,n
          do j = 1,n
            a(j,i) = i
          end do
        end do

! then initialize the B matrix
        do i = 1,n
          do j = 1,n
            b(j,i) = i
          end do
        end do

      end subroutine initialize

      subroutine multiply_matrices(answer, buffer, b, matsize)
        double precision buffer(matsize), answer(matsize)
        double precision b(matsize, matsize)
        integer i, j
! multiply the row with the column
```

Outline | Make ...

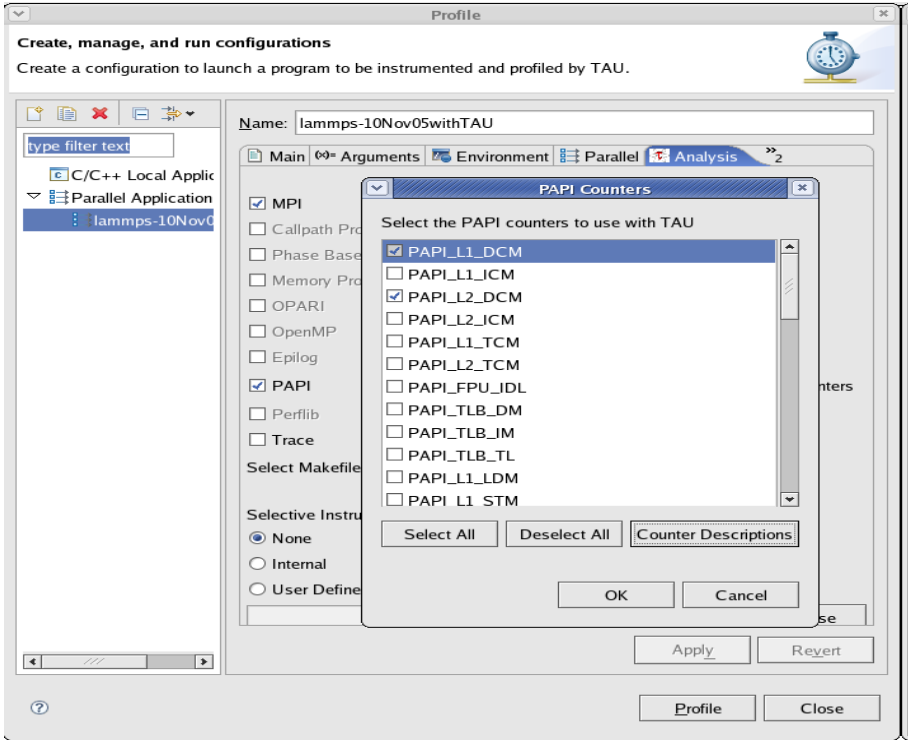
- initialize
- multiply_matrices
- main

Problems | Console | Properties | Performance Data Manager | Launch ParaProf

- AORSA2D
- matmultiply
  - Experiment
    - The New Trial: 2006-12-02 20:36:59
- mm
- ring

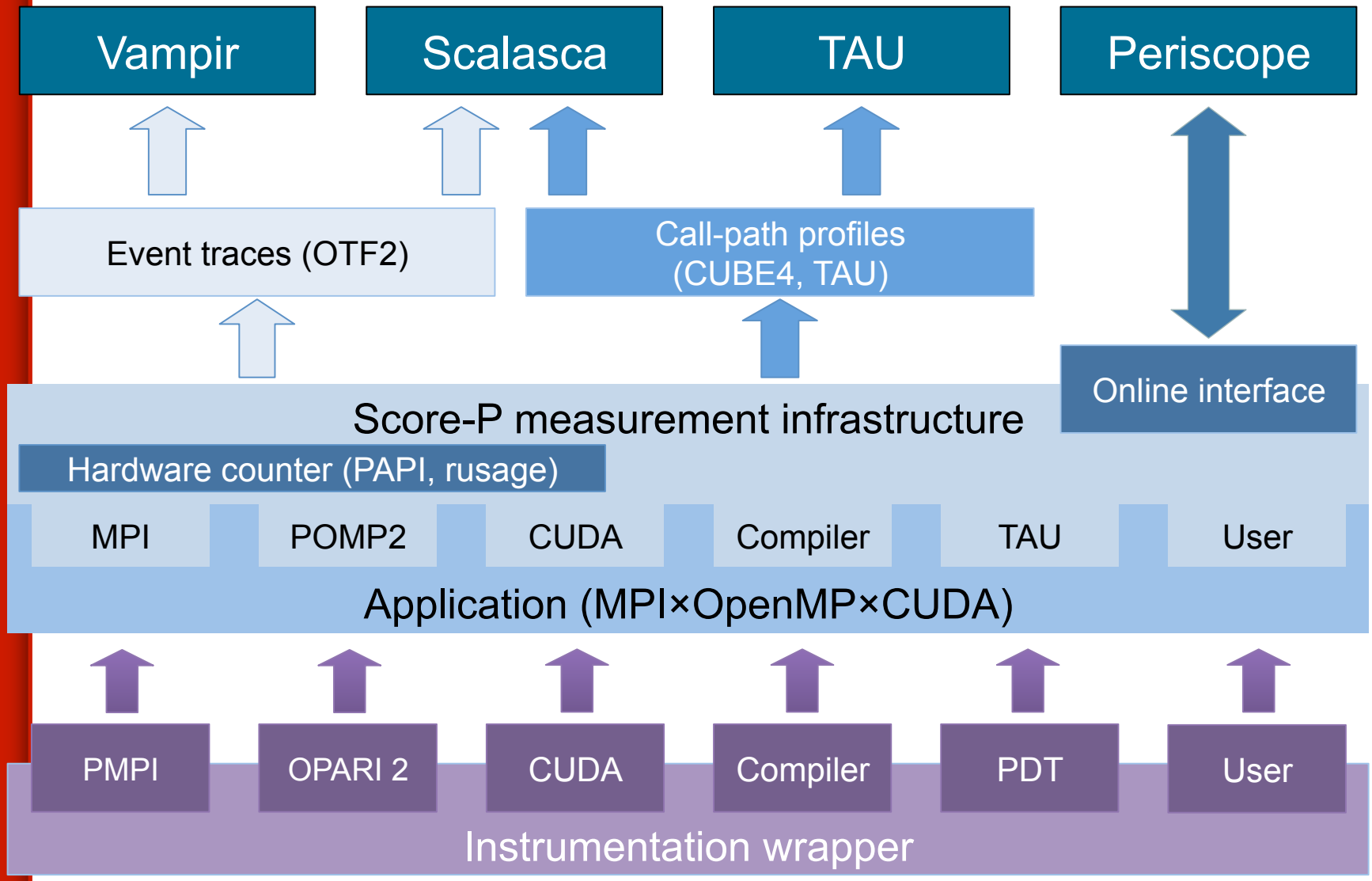
PerfDMF

Choosing PAPI Counters with TAU in Eclipse



Counter	Definition
PAPI_L1_DCM	Level 1 data cache misses
PAPI_L1_ICM	Level 1 instruction cache misses
PAPI_L2_DCM	Level 2 data cache misses
PAPI_L2_ICM	Level 2 instruction cache misses
PAPI_L1_TCM	Level 1 cache misses
PAPI_L2_TCM	Level 2 cache misses
PAPI_FPU_IDL	Cycles floating point units are idle
PAPI_TLB_DM	Data translation lookaside buffer misses
PAPI_TLB_IM	Instruction translation lookaside buffer misses
PAPI_TLB_TL	Total translation lookaside buffer misses
PAPI_L1_LDM	Level 1 load misses
PAPI_L1_STM	Level 1 store misses
PAPI_L2_LDM	Level 2 load misses
PAPI_L2_STM	Level 2 store misses
PAPI_STL_ICY	Cycles with no instruction issue
PAPI_HW_INT	Hardware interrupts
PAPI_BR_TKN	Conditional branch instructions taken
PAPI_BR_MSP	Conditional branch instructions mispredicted
PAPI_TOT_INS	Instructions completed
PAPI_FP_INS	Floating point instructions
PAPI_BR_INS	Branch instructions
PAPI_VEC_INS	Vector/SIMD instructions
PAPI_RES_STL	Cycles stalled on any resource
PAPI_TOT_CYC	Total cycles
PAPI_L1_DCH	Level 1 data cache hits
PAPI_L2_DCH	Level 2 data cache hits
PAPI_L1_DCA	Level 1 data cache accesses
PAPI_L2_DCA	Level 2 data cache accesses
PAPI_L2_DCR	Level 2 data cache reads
PAPI_L2_DCW	Level 2 data cache writes

Score-P Architecture: Common Layer



Partners

Forschungszentrum Jülich, Germany

German Research School for Simulation Sciences, Aachen, Germany

Gesellschaft für numerische Simulation mbH Braunschweig, Germany

RWTH Aachen, Germany

Technische Universität Dresden, Germany

Technische Universität München, Germany

University of Oregon, Eugene, USA

JÜLICH FORSCHUNGSZENTRUM

German Research School for Simulation Sciences

gns

TUM Technische Universität München

RWTH AACHEN UNIVERSITY

TECHNISCHE UNIVERSITÄT DRESDEN

UNIVERSITY OF OREGON

ParaTools

UNIVERSITY OF OREGON

Support Acknowledgments

US Department of Energy (DOE)

- Office of Science contracts
- SciDAC, LBL contracts
- LLNL-LANL-SNL ASC/NNSA contract
- Battelle, PNNL contract
- ANL, ORNL contract

Department of Defense (DoD)

- PETTT, HPCMP

National Science Foundation (NSF)

- Glassbox, SI-2

Partners:

University of Oregon

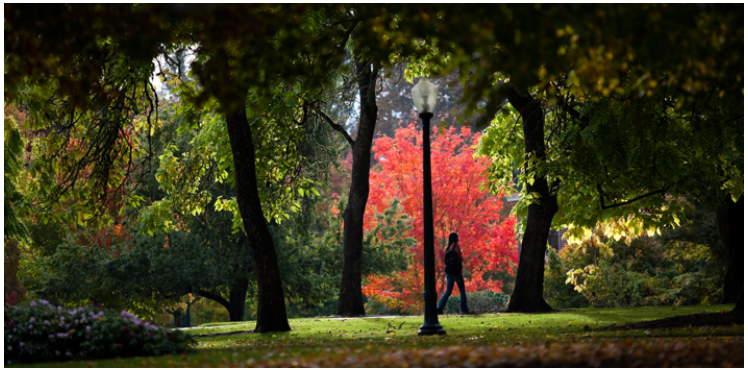
ParaTools, Inc.

University of Tennessee, Knoxville

T.U. Dresden, GWT

Juelich Supercomputing Center

PRL, University of Oregon, Eugene



www.uoregon.edu

Download TAU from U. Oregon.

http://tau.uoregon.edu

http://www.hpclinux.com
[LiveDVD]